MASTER EN SCIENCES INFORMATIQUES

Etude empirique de l'impact du patron architectural MVC et de ses variantes sur la maintenabilité interne des programmes

De Smet, Benoît; Lempereur, Lorent

Award date:
2011

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix de Namur
Faculté d'Informatique
21, Rue Grandgagnage
B-5000 Namur

Année académique 2010-2011

Étude empirique de l'impact du patron architectural MVC et de ses variantes sur la maintenabilité interne des programmes

DE SMET Benoît LEMPEREUR Lorent

Mémoire présenté en vue de l'obtention du diplôme de Master en Sciences Informatiques

Résumé

Quel que soit le cycle de développement logiciel adopté pour un projet, l'activité de maintenance est présente. Cette phase se révèle être la plus coûteuse de chaque projet logiciel. Les développeurs utilisent couramment les patrons architecturaux qui sont reconnus dans la communauté du génie logiciel comme garants de la qualité de maintenabilité. Parmi ces patrons, certains permettent d'architecturer les interfaces utilisateurs. Pour la réalisation d'interfaces utilisateurs interactives, le patron architectural Modèle-Vue-Contrôleur (MVC) est sans nul doute le plus célèbre et le plus couramment utilisé. Plusieurs variantes de ce patron existent tels que Modèle-Vue-Présentateur (MVP) et *Model-Delegate* (MD) que nous présentons et décrivons en détails. Ce mémoire présente une étude empirique réalisée dans le but d'évaluer l'impact de ces trois patrons sur la maintenabilité interne des programmes. Durant cette expérience, des sujets exécutent des tâches de maintenance sur les diagrammes de classe UML de Swing et JFreeChart dans lesquels est injecté aléatoirement un de ces patrons. L'effort demandé lors de l'exécution de ces tâches est mesuré avec trois types de métriques : (1) des métriques évaluant la performance, (2) des métriques physiologiques obtenues à l'aide d'un oculomètre et de la théorie Vision–Compréhension et (3) des mesures subjectives fournies par les sujets. Outre l'analyse des patrons architecturaux, les deux facteurs que sont le niveau d'étude et le sexe sont également pris en compte. Ces analyses ont montré, entre autre, l'influence du niveau d'étude sur la façon d'observer les diagrammes UML. De plus, nous présentons une nouvelle métrique permettant de comparer l'aspect dynamique des parcours visuels des sujets. Cette métrique a permis de montrer que le niveau d'expérience des sujets influe sur la dynamique du parcours visuel. Enfin, le logiciel TAUPE développé afin de supporter l'analyse de données oculométriques est introduit.

Mots clés : génie logiciel empirique, maintenabilité, compréhension de programmes, diagramme de classes UML, oculométrie, patrons architecturaux, mesure de l'effort, MVC.

Abstract

Regardless of the software development cycle adopted for a project, the maintenance activity is always present. This phase is the most expensive for most software project. Developers commonly use architectural patterns that are recognized in the software engineering community to ensure maintainability. Among these patterns, some are used to architecture user interfaces. For interactive user interfaces, the architectural pattern Model-View-Controller (MVC) is undoubtedly the most famous and used pattern. Several variations of this pattern exist, such as Model-View-Presenter (MVP) and Model-Delegate (MD), which we present and describe in details. This master thesis presents an empirical study conducted to evaluate the impact of these three patterns on the internal maintainability of programs. During the experiment, subjects performed maintenance tasks on UML class diagrams of Swing and JFreeChart in which these patterns were randomly injected. The effort required during the execution of these tasks was measured by three types of measurements : (1) performance metrics, (2) physiological metrics obtained using an eye-tracker and the Vision-Comprehension theory, and (3) subjective measures provided by the subjects. In addition to the analysis of architectural patterns, the two factors, study level and gender, are also studied. Theses analyses highlighted, among other things, the influence of the educational level on the way developers look at UML class diagrams. We also introduced a new metric to compare visual paths. This metric showed that the subjects' experience level affects the dynamic of the visual path. Finally, the software TAUPE developed to support the analysis of eye-tracking data is introduced.

Keywords : empirical software engineering, maintainability, program understanding, UML class diagram, eye tracking, architectural patterns, effort assessment, MVC.

Nous tenons à remercier particulièrement notre maître de stage Yann-Gaël Guéhéneuc et notre promoteur Naji Habra pour leur support, leurs relectures, leurs conseils et sans qui notre stage à l'École Polytechnique de Montréal n'aurait pas pu avoir lieu.

Un grand merci à toutes les personnes ayant participé à la relecture de ce document, en particulier Virginie Lempereur et Nicolas Kaczynski pour leurs relectures attentives.

Nous désirons également remercier toutes les personnes ayant accepté de passer notre expérience dont le Ptidej Lab et le SoccerLab pour leur aide.

En outre, nous tenons à remercier toutes les personnes qui nous ont soutenues, de près ou de loin, durant la rédaction de ce document, en particulier nos compagnes pour leur patience et leur compréhension ainsi que nos amis et nos familles pour leur soutien moral.

À tous, merci.

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction

LES cycles de développement logiciel [Som06] sont traditionnellement divisés en plusieurs macro-phases : de l'analyse des besoins à l'implémentation et de la maintenance jusqu'à la fin du projet. Quelque soit le cycle de développement logiciel adopté pour un projet, **l'activité de maintenance** est présente dans chacun d'entre eux. En outre, cette phase se révèle être la plus coûteuse de chaque projet logiciel [Boe02]. Pour les projets d'envergure, un cycle de développement dure généralement plusieurs années [Leh80] et la maintenance est donc rarement effectuée par les développeurs qui ont travaillé en premier sur le programme. Par conséquent, les mainteneurs du projet doivent en premier lieu comprendre le programme avant d'implémenter le moindre changement.

Les développeurs de logiciel utilisent couramment les patrons [Ale79, GHJV94, BMR+96] qui sont reconnus dans la communauté du génie logiciel comme synonymes de qualité. Ces patrons décrivent un problème récurrent de conception qui se pose dans des contextes spécifiques et présentent un schéma générique pour la solution de ce problème. En particulier, les "patrons architecturaux" expriment chacun un schéma structurel, organisationnel et fondamental pour les programmes. Ils fournissent chacun une structure en sous-systèmes, spécifient leurs responsabilités et incluent des règles et des lignes de conduite pour l'organisation de leurs relations [BMR+96].

Au fil du temps, les interfaces utilisateurs des programmes se sont vues améliorées de composants graphiques. À l'heure actuelle, les interfaces graphiques sont présentes dans la majorité des programmes. En outre, de nombreux domaines de l'informatique s'intéressent à ces interfaces, tels que la programmation générative et l'ingénierie dirigée par les modèles [SV04, dFC+11].

Pour la réalisation d'interfaces utilisateurs interactives, le patron architectural Modèle-Vue-Contrôleur est sans nul doute le plus célèbre et utilisé. Néanmoins, plusieurs patrons sont généralement évoqués sous ce terme. En effet, alors que certains projets sont couramment déclarés comme structurés selon ce patron, il s'agit parfois en réalité de variantes, voire d'alternatives telles que le Modèle-Vue-Présentateur ou le *Model-Delegate*. Par exemple, la bibliothèque très célèbre qu'est Java Swing déclare être architecturée selon le patron Modèle-Vue-Contrôleur alors que sa structure relève en réalité du patron *Model-Delegate*.

1.1 Problématique

La maintenabilité se révèle donc être un enjeu majeur dans le domaine du génie logiciel. De plus, une maintenabilité aisée est une qualité recherchée non seulement à chaque étape des processus de développement, mais également dans l'intégralité des produits logiciels. Dans un monde du logiciel

où l'utilisateur final prend de plus en plus de place dans le processus (notamment dans les approches de développement *Agile* [BBvB+01], par exemple), l'interface utilisateur est couramment soumise au changement. En outre, ces interfaces forment l'unique relation que l'utilisateur entretient avec le programme. Par conséquent, les interfaces utilisateurs, graphiques ou non, sont indéniablement soumises au respect de cette exigence de maintenabilité.

Selon la littérature, les patrons architecturaux semblent garantir des qualités telles que la flexibilité et la compréhensibilité [Wen01]. Cependant, peu d'études empiriques existent et confirment de telles évidences. Toutefois, le génie logiciel tente de combler les fossés entre la théorie et la pratique. Les études sur les patrons architecturaux des interfaces utilisateurs sont peu nombreuses (par exemple, [DS99]) et portent généralement sur un patron particulier, mais ne les comparent pas entre eux.

Il est important que les développeurs aient connaissance des outils qui sont à leur disposition et de leurs impacts. La difficulté d'apprentissage de certains patrons peut même empirer la situation. En effet, plusieurs observations en industrie montrent que la mauvaise utilisation ou la sur-utilisation des patrons peut impliquer l'effet inverse de celui désiré, à savoir une bonne maintenabilité [Wen01].

La confusion entre le patron Modèle-Vue-Contrôleur et ses variantes remet en cause son utilisation aveugle et sans contrôle. Néanmoins, à notre connaissance, aucune étude ne confronte ces différents patrons architecturaux entre eux. L'importance de bien connaître ces différentes alternatives et de ne pas se limiter à un seul patron architectural est d'apporter une solution appropriée à un problème qui se présente.

De ce constat, il est possible de mettre en évidence plusieurs problèmes :

- Le **vocabulaire** précis quant aux patrons architecturaux relatifs aux interfaces utilisateurs est **disparate** et **mal connu**.
- La maintenabilité occasionnée par ces patrons est considérable au vu de l'omniprésence des interfaces utilisateurs. Néanmoins, **aucune comparaison**, à notre connaissance, n'existe **entre eux**.
- Le génie logiciel dispose de **peu d'information** quant à la relation **entre la maintenabilité** et **ces patrons**. À l'heure actuelle, il s'agit tout au plus d'observations circonstancielles et de retours d'expérience.

1.2 Questions de recherche

L'*analyse de l'impact du patron architectural Modèle-Vue-Contrôleur et de ses variantes sur la maintenabilité interne des programmes* soulève diverses questions qui font l'objet de ce mémoire. Tout au long du document, nous tentons d'apporter des réponses à ces questions et détaillons notre approche :

Q1 Comment définir le terme de "maintenabilité" ?

Q2 Quelles théories et outils le génie logiciel offre t-il pour mesurer la maintenabilité d'un programme ? Ces approches sont-elles suffisantes ? D'autres domaines peuvent-ils apporter une aide à ces mesures ?

Q3 Comment réaliser une approche empirique pour évaluer cette qualité ? Comment y inclure ces théories ?

Q4 Les outils disponibles sont-ils suffisants pour supporter ces théories ?

Q5 Quelles alternatives et variantes existe t-il au patron Modèle-Vue-Contrôleur et comment sont-elles définies ?

Q6 Ces différents patrons architecturaux ont-ils un impact différent sur la maintenabilité ?

Q7 Quelles caractéristiques des mainteneurs peut influencer cet impact ?

1.3 Approche et méthodologie

Ce mémoire présente une expérimentation réalisée avec l'objectif d'**étudier empiriquement l'impact du patron MVC et de deux de ses alternatives sur la maintenabilité d'un programme**.

Pour ce faire, des sujets expérimentés en programmation orienté-objet doivent effectuer plusieurs tâches de maintenance sur différents programmes représentés sous forme de diagrammes de classes UML [Obj09]. Face aux tâches exécutées par les sujets, trois types d'analyse sont opérés afin de quantifier leur réalisation. Premièrement, nous effectuons une approche d'étude de la performance basée sur le temps et la qualité des solutions proposées par les sujets. Ensuite, une étude sur la compréhension des diagrammes par les sujets est réalisée à l'aide de la théorie Vision–Compréhension. Enfin, une approche subjective permet au sujet d'exprimer sa perception quant à l'effort qu'il a fourni durant ses tâches.

La théorie de la compréhension tente d'aider de telles expériences. Elle a été étudiée par de nombreux chercheurs, par exemple pour construire des modèles de compréhension de programmes, tels que [Bro78, vMV95, SFM99], ou des outils pour faciliter cette activité,par exemple [BG97]). Récemment, des chercheurs ont introduit l'utilisation d'appareils d'oculométrie pour améliorer notre compréhension des processus cognitifs des développeurs qui prennent place lors de la compréhension des programmes [Gué06, YKM07, Gué09]. Cette utilisation de l'oculométrie a été, par exemple, appliquée à l'étude de données visualisées telles qu'un code source, des identificateurs [SM10], des diagrammes [CG10], etc. En particulier, Guéhéneuc [Gué09] propose une théorie pour unifier celles de la compréhension de programmes avec les théories actuelles en science de la vision [Pal99].

Les appareils d'oculométrie ont déjà été utilisés durant les 30 dernières années en science cognitive pour étudier les interfaces homme/machine, pour le marketing, les recherches médicales, etc. Un oculomètre enregistre les coordonnées du regard d'un sujet lorsqu'il regarde un écran. Il fournit une nouvelle perspective sur le processus de compréhension d'un sujet car il montre les zones qui attirent l'attention de celui-ci ainsi que le parcours visuel du regard sur l'écran [Duc07].

Les métriques existantes dans le domaine de l'oculométrie n'étant pas toujours suffisantes, l'introduction de nouvelles métriques est nécessaire. Un problème similaire est présent pour les logiciels qui supportent les processus expérimentaux dans ce domaine, qu'il s'agisse d'acquisition d'information ou de traitement des données. En effet, les logiciels existants sont généralement inextensibles et limitent donc l'évolution du domaine en question par des chercheurs ne disposant pas des droits nécessaires sur ces programmes.

1.4 Structure du document

La première partie présente les **différents domaines et matériaux** nécessaires à la compréhension de ce mémoire. Tout d'abord, le Chapitre 2 définit l'objet de notre étude, c'est-à-dire les patrons de conception et plus spécifiquement les patrons architecturaux liés aux interfaces utilisateurs. Cet

objet d'étude est analysé selon le critère de maintenabilité. Le Chapitre 3 se charge de définir plus formellement ce critère dans le cadre de la qualité des produits logiciels. L'activité de maintenance implique nombre de processus cognitifs qui permettent aux ingénieurs du logiciel d'appréhender les programmes. Les théories de compréhension, de vision et de Vision–Compréhension, énoncées et détaillées au Chapitre 4, tentent d'étendre notre connaissance afin de comprendre ces processus. Outre ces théories, la qualité de maintenabilité d'un programme peut être corrélée à l'effort fourni par un développeur pour mener à bien une tâche de maintenance. Les techniques permettant de mesurer cet effort sont brièvement expliquées au Chapitre 5. Afin d'être étudiées, ces tâches doivent être mises dans un contexte particulier. Notre approche repose sur une étude empirique, dans le cadre du génie logiciel, dont la structure est décrite au Chapitre 6. Des éléments mathématiques et plus particulièrement statistiques sont nécessaires à la compréhension des analyses réalisées pour interpréter les résultats de l'expérience proposée dans ce travail. Ces bases sont rapidement expliquées au Chapitre 7. L'étude empirique en question repose sur l'utilisation de différents "instruments", logiciels et matériels, pré-existants ou non, qui sont exposés au Chapitre 8.

La deuxième partie est principalement consacrée à notre **étude empirique**. Alors qu'il existe plusieurs métriques dans les différents domaines touchant à notre problématique, une nouvelle métrique proposée pour l'occasion est décrite au Chapitre 9 dans le cadre de l'analyse dynamique de parcours visuel. Elle est préalablement décrite formellement et ensuite appliquée à une étude de cas réelle. Cette métrique est ensuite utilisée lors de l'analyse des résultats de notre étude. Dans le cadre de cette étude empirique, le Chapitre 10 détaille les éléments constitutifs de l'expérience. Il contient la description et la justification des choix effectués sur les différents éléments y prenant part. Ensuite, le Chapitre 11 explique la configuration de l'expérience, son déroulement et ses aspects éthiques. Une fois l'intégralité de l'expérience terminée, les données recueillies sont analysées avant d'en tirer des conclusions. Trois analyses sont opérées sur les données : une analyse de la performance, des mesures oculométriques et des mesures subjectives respectivement aux Chapitres 12, 13 et 14. Le Chapitre 15 fait une rétrospective de l'étude dans son intégralité, c'est-à-dire les menaces à la validité et les leçons tirées d'une telle expérience. Enfin, le Chapitre 16 tire une conclusion de ce travail, met en avant notre contribution dans différents domaines, qu'il s'agisse de résultats d'expériences ou d'outils mis en place et propose différentes perspectives à explorer par la suite.

1.5 Limites du mémoire

Notre étude comporte des limites qu'il est nécessaire d'énoncer afin d'avertir le lecteur de ce qu'il pourra trouver, ou non, au sein de ce document. Ces limites peuvent parfois porter atteinte à la validité de nos conclusions mais sont nécessaires car dans de telles études, les compromis sont inévitables. Cette section n'a pas pour objectif d'énoncer des menaces à la validité de notre travail, celles-ci étant détaillées en Section 15.2.

L1 Si le document présente des alternatives et variantes du Modèle-Vue-Contrôleur, un sous-ensemble d'entre eux seulement fait l'objet de l'étude expérimentale. La présentation préliminaire de patrons, qu'ils soient inclus dans l'étude ou non, permet de fixer un vocabulaire réduisant les ambiguïtés à leur sujet. Par exemple, le patron architectural Présentation-Abstraction-Contrôle (PAC) [BMR+96] n'est pas pris en compte dans ce document.

L2 Tous les sujets de l'expérience travaillent dans le domaine académique. L'expérience n'est donc pas représentative de l'industrie.

L3 Bien que la maintenabilité sera définie plus loin dans ce document, la portée de cette définition ne tient pas compte de certains de ses sous-critères de qualité. Ce choix sera justifié mais mérite

tout de même une place dans les limites. Par exemple, la testabilité des patrons architecturaux n'est pas évaluée.

L4 La maintenabilité est évaluée en terme de qualité de produit logiciel et ne concerne donc pas la qualité des processus.

L5 L'étude est empirique et souffre donc des inconvénients inhérents à cette approche.

L6 L'approche choisie ne considère que le mainteneur comme acteur du processus. Les autres parties prenantes ont été ignorées.

Première partie

Contexte

CHAPITRE 2

Patrons architecturaux

> Good programmers use their brains, but good guidelines save us having to think out every case.
>
> Francis Glassborow

LA problématique décrite dans ce document pose les patrons architecturaux comme objet de l'étude empirique. Ce chapitre a pour objectif de présenter préalablement les patrons dans un cadre général en Section 2.1, de les définir et de les introduire dans le contexte du génie logiciel. Les qualités qui sont attendues de leur utilisation sont décrites en Section 2.2. Ensuite, la Section 2.4 présente plus spécifiquement les patrons architecturaux relatifs aux interfaces utilisateurs. L'accent est principalement mis sur le patron Modèle-Vue-Contrôleur pour ensuite décrire plusieurs de ses alternatives en Section 2.5.

La Section 2.1, 2.2, 2.3 sont inspirées de Buschmann et al. [BMR+96] et de de Gamma et al. [GHJV94].

Sommaire

2.1 Patrons de conception

Les patrons de conception aident le développeur sur base de l'expérience collective des ingénieurs du logiciel. Ils capturent une expérience existante et éprouvée en développement de logiciel et aident à promouvoir les bonnes pratiques de conception [1]. Chaque patron traite d'un problème spécifique et récurrent dans la conception et l'implémentation de systèmes logiciels. Ils peuvent être utilisés pour construire des architectures logicielles avec des propriétés spécifiques [BMR+96].

2.1.1 Histoire

Le terme de patron, traduction française de *pattern*, est introduit dans le domaine de l'ingénierie du logiciel en 1987 [BC87] par Beck et Ward [Smi87]. Ce concept est initialement utilisé dans le domaine de l'architecture par Alexander [Ale79].

Alors que Gamma et al. [2] réalisent un travail considérable en mettant en place un catalogue de 23 patrons de conception en 1994 [GHJV94], Buschmann et al. en apportent une autre perspective dans [BMR+96] en les étudiant selon trois niveaux (les patrons architecturaux, les patrons de conception et les idiomes). Cette section a pour objectif de décrire les patrons de conception au travers de ces différentes perspectives et de tenter de les concilier.

2.1.2 Contexte

Lorsque des experts travaillent sur un problème particulier, il est inhabituel pour eux de l'aborder en inventant une nouvelle solution qui est complètement distincte de celles déjà existantes. Ils rencontrent souvent des problèmes similaires à ceux qu'ils ont déjà rencontrés et ils réutilisent l'essence de cette solution pour résoudre le nouveau problème. Le type de "comportement d'expert", cette pensée de paire *problème–solution*, est commune à plusieurs domaines différents tels que l'architecture [Ale79], l'économie et le génie logiciel [BJ94]. Il s'agit d'une manière naturelle de faire face à chaque type de problème. Selon Johnson, un des auteurs de [GHJV94] :

> "Ces paires problème–solution tendent à se regrouper en famille de problèmes et de solutions similaires avec chacune d'entre elle exposant un patron aussi bien dans les problèmes que dans les solutions " [BMR+96, p. 3]

2.1.3 Définition préliminaire

Alexander [Ale79], l'auteur initial du terme *pattern* dans le domaine de l'architecture, le définit comme :

> "Chaque patron est une règle en trois parties qui exprime une relation entre un certain contexte, un problème et une solution.
>
> Tel un élément dans le monde, chaque patron est une relation entre un contexte, un certain système de forces qui survient à plusieurs reprises dans ce contexte et une certaine configuration spatiale qui permet à ces forces de se résoudre par elles-mêmes.
>
> Tel un élément du langage, un patron est une introduction qui montre comment cette configuration spatiale peut être utilisée, tant et plus, pour résoudre le système de forces

1. Tout au long du document, le terme anglais "*design*" sera traduit par "conception".
2. Nommés ensuite "*Gang of Four*".

donné, partout où le contexte le rend pertinent.

> Le patron est, en bref, à la fois une chose qui se produit dans le monde, une règle qui nous raconte comment créer cette chose et quand nous devons la créer. C'est à la fois un traitement et une chose ; une description d'une chose qui est en vie et une description du processus qui générera cette chose. " [Ale79, p. 246]

Cette définition s'accorde bien avec le domaine de l'architecture. Néanmoins, les patrons se retrouvent également dans les architectures logicielles. Les experts en génie logiciel connaissent ces patrons d'expérience pratique et les suivent pour développer des programmes ayant des propriétés spécifiques. Ils les utilisent pour résoudre les problèmes de conception de manière effective et élégante [BMR+96].

2.1.4 Propriétés

Plusieurs propriétés sont à extraire des patrons en architecture logicielle. Elles permettent d'atteindre une définition plus adaptée au génie logiciel.

Propriété 1 *Un patron s'adresse à un problème de conception récurrent qui survient dans une situation de conception spécifique et présente une solution.* Cette propriété s'approche de la définition énoncée par Prechelt et al. :

> "Les patrons de conception logiciels rassemblent des solutions éprouvées à des problèmes de conception récurrents dans une forme qui en simplifie la réutilisation. " [PUT+01]

Propriété 2 *Les documents sur les patrons existent et présentent des conceptions éprouvées par l'expérience.* Briand et al. utilisent cette propriété pour définir les patrons dans leurs travaux.

> "Les patrons de conception logiciels sont des bonnes pratiques documentées qui peuvent être appliquées à des problèmes récurrents. " [BLS06]

Propriété 3 *Les patrons identifient et spécifient des abstractions qui sont au-dessus du niveau de simples classes et d'instances, ou de composants [GHJV94].* Typiquement, un patron décrit plusieurs composants, classes et objets et détaille leurs responsabilités et leurs relations. Tous les composants, ensemble, résolvent le problème auquel le patron s'adresse et généralement plus efficacement qu'un simple composant.

Propriété 4 *Les patrons fournissent un vocabulaire commun et une compréhension des principes de conception [GHJV94].* Les noms des patrons, s'ils sont choisis avec attention, deviennent part d'un langage de conception répandu. Ils facilitent l'efficacité des discussions au sujet des problèmes de conception et de leurs solutions. Ils suppriment la nécessité d'expliquer une solution à un problème particulier avec une explication longue et compliquée.

Propriété 5 *Les patrons sont un moyen de documenter les architectures logicielles.* Ils permettent de décrire la vision que nous avons lorsque nous concevons un système logiciel. Cela aide les autres concepteurs à éviter la violation de cette vision lorsqu'ils étendent et modifient l'architecture originale ou le code du système.

Propriété 6 *Les patrons supportent la construction de logiciels avec des propriétés définies.* Les patrons fournissent un squelette de comportement fonctionnel et par conséquent, ils aident à implémenter les fonctionnalités de l'application. En outre, les patrons abordent explicitement les exigences

non-fonctionnelles de systèmes logiciels, telles que la changeabilité, la testabilité et la réutilisabilité. Cette perception est notamment proposée par Noda et al. [NK01] et Aversano et al. [ACC+07].

Propriété 7 *Les patrons aident à la construction d'architectures logicielles complexes et hétérogènes.* Chaque patron fournit un ensemble prédéfini de composants, de rôles et de relations entre eux. Ils peuvent être utilisés pour spécifier des aspects particuliers de structures concrètes de logiciels. Néanmoins, lorsqu'un patron détermine la structure de base d'une solution, il ne spécifie pas une solution pleinement détaillée. Un patron fournit un schéma pour une solution générique à une famille de problème plutôt qu'un module préfabriqué qui peut être utilisé tel quel.

Propriété 8 *Les patrons aident à la gestion de la complexité du logiciel.* Chaque patron décrit une façon éprouvée de gérer le problème auquel il s'adresse [BMR+96].

Malgré ces propriétés, des critiques peuvent être formulées. Par exemple, Wendorff énonce dans [Wen01, p. 2] :

> "*U*n patron n'est pas nécessairement une bonne pratique, il s'agit plutôt d'une approche de praticien éprouvée. Par conséquent, un patron est simplement une option de conception parmi d'autres durant le développement d'un logiciel. La décision de favoriser l'une ou l'autre option devrait normalement être basée sur les objectifs détaillés et spécifiques du projet logiciel. "

2.1.5 Définition finale considérée

Sur base de ces propriétés, Buschman et al. [BMR+96] définissent un patron dans le cadre de l'Ingénierie du logiciel tel que :

> "*U*n patron en architecture logicielle décrit un problème particulier récurrent de conception qui se pose dans des contextes spécifiques de conception et présente un schéma générique éprouvé pour sa solution. Le schéma de solution est spécifié par la description de ses composants constitutifs, leurs responsabilités, leurs relations et la manière dont ils collaborent. "

2.1.6 Constitution

Cette nouvelle définition mentionne les éléments constitutifs d'un patron. Quels sont-ils ? Alors que Buschamnn et al. déterminent trois éléments, Gamma et al. en définissent quatre. En effet, Gamma et al. énoncent le nom, le problème, la solution et les conséquences. Buschamnn et al., eux, détaillent le contexte, le problème et la solution. Le concept de "problème" de Gamma et al. englobe en réalité le contexte et le problème selon Buschmann et al.. En considérant ces nuances, nous pouvons affirmer que chaque patron sous-tend un schéma en cinq parties :

1. Le **nom** du patron est un élément utilisable pour décrire un problème de conception, ses solutions et ses conséquences en un mot ou deux. Nommer un patron augmente immédiatement notre conception du vocabulaire. Trouver de bons noms a été, selon Gamma et al., une des étapes les plus difficiles dans le développement du catalogue de [GHJV94].
2. Le **contexte** décrit une situation donnée qui soulève un problème. Il étend la dichotomie *problème–solution* en décrivant les situations dans lesquelles le problème survient [BMR+96].

3. Le **problème** décrit quand appliquer un patron. Il explique le problème récurrent soulevé à répétition par le contexte donné. Il commence avec une spécification générale du problème, en capturant sa véritable essence [BMR+96]. Parfois, le problème peut inclure une liste de conditions qui doivent être rencontrées pour donner du sens à l'application d'un patron. La représentation d'algorithmes sous forme d'objets en est un exemple [GHJV94].
4. La **solution** décrit les éléments qui forment la conception, leurs relations, leurs responsabilités et leur collaboration [GHJV94]. Cette solution montre comment résoudre le problème récurrent, ou mieux, comment équilibrer les forces qui lui sont associées [BMR+96].
5. Les **conséquences** sont le résultat de négociations lors de l'application d'un patron. Ces conséquences sont rarement exprimées lorsque des décisions de conception sont décrites. Ils sont critiques pour l'évaluation des alternatives et la compréhension des coûts et des bénéfices de l'application d'un patron. Les conséquences sur un logiciel se préoccupent souvent des compromis entre temps et espace, c'est-à-dire entre performance et mémoire. Elles peuvent également s'adresser à des problèmes d'implémentation et de langage. Alors que la réutilisation est souvent un facteur en conception orientée objets, les conséquences d'un patron incluent son impact sur la flexibilité d'un système, sur son extensibilité, ou sa portabilité. Lister explicitement ces conséquences nous aide à les comprendre et à les évaluer.

Chaque point de vue affecte l'interprétation de ce qu'est ou n'est pas un patron. Le patron d'une personne peut être un bloc primitif de construction pour une autre personne. Gamma et al. se sont concentré sur les patrons d'un certain niveau d'abstraction [GHJV94]. Afin de comprendre le rôle des patrons dans la compréhension de programmes, il est important de garder à l'esprit qu'un patron est une construction mentale [BMR+96].

2.1.7 Catégories

La catégorisation des patrons est considérée différemment selon Gamma et al. ou selon Buschamnn et al.. Ces premiers [GHJV94] classifient les patrons selon deux critères : le **but** (*purpose*), qui reflète ce qu'un patron fait et la **portée** (*scope*) qui spécifie si un patron s'applique principalement aux classes ou aux objets. Selon leur but, ils déterminent trois familles de patrons :

1. Les patrons **créationnels** concernent le processus de création des objets.
2. Les patrons **structurels** traitent avec la composition de classes ou d'objets.
3. Les patrons **comportementaux** caractérisent les méthodes avec lesquelles les classes ou les objets interagissent et distribuent la responsabilité.

Le critère de portée divise les patrons en deux catégories :

1. Les patrons de **classe** traitent des relations entre les classes et leurs sous-classes. Ces relations sont réalisées à l'aide du mécanisme d'héritage et sont donc statiques et fixées au moment de la compilation.
2. Les patrons d'**objet** traitent des relations entre les objets, ce qui peut changer à l'exécution et sont donc plus dynamiques.

Selon Buschamnn et al. [BMR+96], il apparaît que les patrons couvrent des gammes d'échelle et d'abstraction différentes. Des patrons aident à structurer un programme en sous-programmes. D'autres patrons supportent le raffinement des sous-programmes et des composants ou des relations entre eux. D'autres encore aident à l'implémentation d'aspects particuliers dans un langage de programmation spécifique. La classification peut se faire selon trois catégories :

1. Un **patron architectural** exprime un schéma structurel, organisationnel et fondamental pour les programmes. Il fournit un ensemble de sous-programmes, spécifie leurs responsabilités et inclut des règles et des lignes de conduite pour l'organisation des relations entre eux.
2. Un **patron de conception** fournit un schéma pour le raffinement des sous-programmes ou des composants d'un programme ou des relations entre eux. Il décrit une structure souvent récurrente de communication de composants qui résout un problème général de conception à l'intérieur d'un contexte particulier [GHJV94].
3. Un idiome est un patron de bas niveau spécifique à un langage de programmation. Il décrit comment implémenter les aspects des composants ou des relations entre eux en utilisant les caractéristiques d'un langage donné.

2.2 Qualité attendue de l'usage des patrons de conception

Les patrons génèrent de la valeur qui dépend de l'utilité des attributs de qualité qui peuvent être atteints par l'application de ces patrons. Néanmoins, il n'existe pas de méthode rigoureuse qui permette d'évaluer la valeur ajoutée des patrons [OKK07]. En outre, ni Gamma et al. [GHJV94] ni Buschamnn et al. [BMR+96] ne mentionnent de modèle de qualité applicable aux patrons de conception. Selon Wendorff, les auteurs qui mentionnent le terme "qualité" se réfèrent généralement à des attributs de qualité tels que la "flexibilité", la "compréhensibilité", la "maintenabilité", etc. sans donner de définition opérationnelle claire de ces concepts [Wen01].

Par conséquent, la décision en faveur ou contre l'utilisation d'un patron reste aujourd'hui une question de jugement subjectif pour au moins deux raisons [Wen01]. Premièrement, notre habilité à mesurer les effets des patrons sur la qualité logicielle est très limitée [PJCK97]. Deuxièmement, il n'existe aucune théorie formelle et solide qui lie les patrons aux concepts de qualité logicielle et en conséquence, il n'est pas possible d'avoir un modèle explicite de comment les patrons fonctionnent (comment les facteurs de succès interagissent, etc.) [Wen01, PUT+01]. Hsueh et al. tentent d'établir une approche quantitative basée sur les modèles de qualité orientés objets pour évaluer la qualité des patrons de conception. Cette approche a pour objectif de vérifier la cohérence entre l'intention du patron et sa structure [HCC08].

Beck et al. décrivent leur expérience de l'utilisation des patrons de conception dans l'industrie [BCM+96]. Dans la même optique, Wendorff décrit un vaste projet commercial où l'utilisation incontrôlée de patrons a contribué à de graves problèmes de maintenance [Wen01]. De son expérience du projet commercial qu'il présente, Wendorff ne tente pas d'apporter une critique négative sur les patrons, mais bien de montrer qu'une utilisation inappropriée des patrons peut provoquer des problèmes. Il catégorise deux types de patrons appliqués d'une façon inappropriée [Wen01, p. 83] :

1. Les patrons qui sont simplement utilisés de façon inadéquate par les développeurs qui n'ont pas compris le raisonnement derrière les patrons.
2. Les patrons qui ne tombent pas dans la première catégorie mais qui ne **correspondent pas aux exigences** du projet. L'analyse des patrons par Wendorff identifie plusieurs situations qui mènent à cette deuxième catégorie.
   - Plusieurs développeurs surestiment la volatilité future des exigences et optent généralement pour des patrons afin de rendre le logiciel plus flexible.
   - Les exigences changent au fil du temps et les patrons qui ont été raisonnables au début deviennent obsolètes ensuite.

- Dans certains cas, les patrons sont appliqués sans aucun regard sur les objectifs de qualité du projet.
- Certains patrons sont utilisés pour leur notoriété, pas parce qu'ils sont réellement nécessaires au projet.

D'autres chercheurs conduisent plutôt des études empiriques dans le but d'en déduire des faits et des théories.

2.2.1 Études empiriques de l'impact de l'usage des patrons

De nombreuses études empiriques ont été réalisées afin d'évaluer la qualité des patrons de conception. Ci-dessous se trouvent plusieurs d'entre elles présentées par ordre chronologique [3].

Prechelt et al. cherchent les évidences des bénéfices qu'apporte l'utilisation des patrons de conception au niveau de la maintenance. Leur expérience (1) examine des scénarios de maintenance de logiciel qui emploient des patrons variés et (2) compare les conceptions utilisant les patrons avec des alternatives plus simples (sans patron). Globalement, l'étude montre que les patrons de conception sont bénéfiques. Les résultats montrent également que l'utilisation des patrons de conception doit généralement être faite à moins qu'il n'y ait une raison particulière d'utiliser une solution plus simple. Néanmoins, contrairement aux croyances communes, les effets bénéfiques des patrons ne sont pas une évidence universelle [PUT+01].

Prechelt et al., au travers d'une étude empirique, se focalisent sur la question "Est-ce que cela aide le mainteneur si le patron de conception dans le code du programme est documenté *explicitement* en comparaison d'un programme bien documenté sans référence explique au patron ?" La technique considérée ici pour documenter les patrons est un bloc de commentaires dans le code. Les sujets effectuent des tâches de maintenance sur deux programmes Java de respectivement 360 et 560 lignes de code (LOC), commentaires inclus. Les deux programmes contiennent des patrons de conception. La configuration de l'expérience n'a pas permis de tirer de résultats, mais les auteurs concluent qu'en fonction du programme et de la tâche à réaliser, de la connaissance du patron et du personnel, les lignes de commentaires dans un programme peuvent réduire considérablement le temps requis pour la modification d'un programme ou peut aider à l'amélioration de la qualité du changement. Ils recommandent donc de toujours documenter explicitement les patrons dans le code source [PULPT02].

Aversano et al. étudient l'évolution des patrons de conception dans trois projets Java : JHotDraw, ArgoUML et Eclipse-JDT. Cette étude analyse à quelle fréquence les patrons sont modifiés, à quels changements ils sont soumis et quelles classes changent en même temps que le patron. Les résultats indiquent que la fréquence des patrons de conception et la quantité de "co-modifications" ne dépendent pas du type de patron, mais plutôt du rôle joué par le patron pour supporter les caractéristiques de l'application [ACC+07].

Di Penta et al. analysent en premier lieu la prédisposition des patrons de conception aux modifications et les types de changements qui surviennent sur les classes jouant un rôle dans ces patrons. Ils présentent une étude empirique afin de comprendre s'il existe des rôles qui sont plus prédisposés aux modifications que d'autres et s'il existe des modifications qui sont plus susceptibles de survenir dans certains rôles. Cette étude porte sur 12 patrons de conception à partir des codes sources de trois

3. Cette section ne présente pas les études de patrons liées à l'analyse de mesures oculométriques.

systèmes différents : JHotDraw, Xerces et Eclipse-JDT. Sur la majorité des patrons étudiés, les résultats suggèrent de concevoir soigneusement les rôles les plus enclins aux changements [DCGA08].

2.2.2 Avantages

Il résulte de certains travaux et certaines études des avantages des patrons de conception. Alors que des avantages sont liés à l'observation des patrons de conception dans l'industrie, d'autres avantages sont tirés d'études empiriques. Ces avantages sont notamment ceux résumés dans [Jea08]. De plus, l'article de Cline [Cli96] fournit une liste de plusieurs avantages et inconvénients des patrons.

Bon médium de communication en équipe Lorsque plusieurs développeurs discutent entre eux, ils utilisent les noms des patrons comme une méthode précise et concise de communiquer des concepts complexes efficacement [BCM+96, Cli96].

Forme compacte La forme compacte que représente un patron de conception permet de capturer les parties essentielles d'une conception. Cette forme aide les développeurs et les mainteneurs à comprendre l'architecture [BCM+96].

Encouragement de la réutilisation des "bonnes pratiques" Ces solutions de conception aident les développeurs les moins expérimentés à concevoir les systèmes [BCM+96].

Réutilisabilité et maintenabilité accrue [ACC+07, p. 1].

Productivité augmentée [PULPT02, p. 1].

Coordination du processus et de la communauté Un vocabulaire commun est fourni par les patrons de conception [GHJV94, Cli96].

Passer d'un compromis à une situation *gagnante–gagnante* Souvent, les concepteurs doivent faire un choix entre deux qualités de haute priorité. Certains des patrons de conception les plus utiles permettent de travailler sans se soucier de ces compromis [Cli96, p. 2].

Sans les citer tous, chaque patron tente d'améliorer la maintenabilité des systèmes.

2.2.3 Inconvénients

Bien que souvent moins traités dans la littérature, certains inconvénients sont liés à l'application des patrons de conception.

Complexité et consommation de temps L'écriture de bons patrons est une compétence qui ne s'acquière pas facilement [BCM+96].

Conception dépendante des patrons Certains patrons qui ont pour intention d'améliorer la "flexibilité" se basent sur le mécanisme d'héritage (par exemple, le patron Observateur). Cela rend les classes de l'application dépendantes des patrons et réduit la réutilisabilité de ces classes [NK01].

Patrons surestimés La popularité de l'approche des patrons de conception empêche parfois les développeurs de percevoir les limites que cette approche pose [Cli96, p. 3].

Apprentissage parfois difficile Certains patrons de conception sont simplement trop difficiles à apprendre pour un concepteur orienté objet moyen [Cli96, p. 3][Wen01].

Classification inutile Les classifications des patrons sont utiles pour les personnes qui comprennent déjà ces patrons, mais lors de leur apprentissage, les modèles mentaux des concepteurs moyens ne correspondent pas forcément aux critères de catégorisation utilisés [Cli96, p. 3].

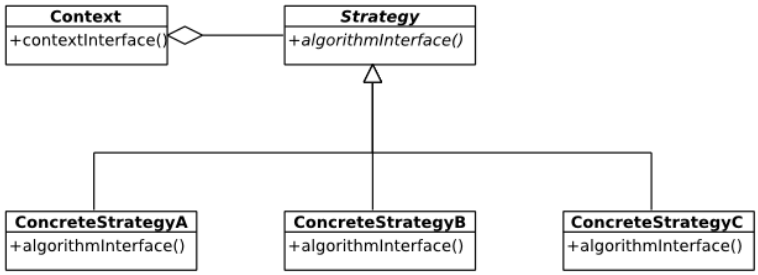


FIGURE 2.1 – Le patron Stratégie tel que défini par Gamma et al. [GHJV94]

À l'instar des avantages, tous les inconvénients potentiels des patrons ne sont pas listés car il ne s'agit pas de l'objectif de ce document. La liste présentée ci-dessus n'a pour objectif que de faire prendre conscience au lecteur de certaines informations qui ne sont que trop peu présentes dans la littérature.

2.3 Exemples de patrons de conception

Cette section présente deux patrons de conception dont la compréhension est nécessaire de la part du lecteur pour la suite du chapitre. Ces deux patrons sont "Stratégie" et "Observateur".

2.3.1 Patron de conception Stratégie

Une "Stratégie" est un objet qui représente un algorithme, qui définit une famille d'algorithmes, encapsule chacun d'entre eux et les rend interchangeables [GHJV94]. La Figure 2.1 représente ce patron de conception où un type `Context` joue le rôle du contexte d'utilisation. Une classe abstraite `Strategy` dispose de plusieurs implémentations différentes. Selon le contexte, une implémentation sera utilisée plutôt qu'une autre.

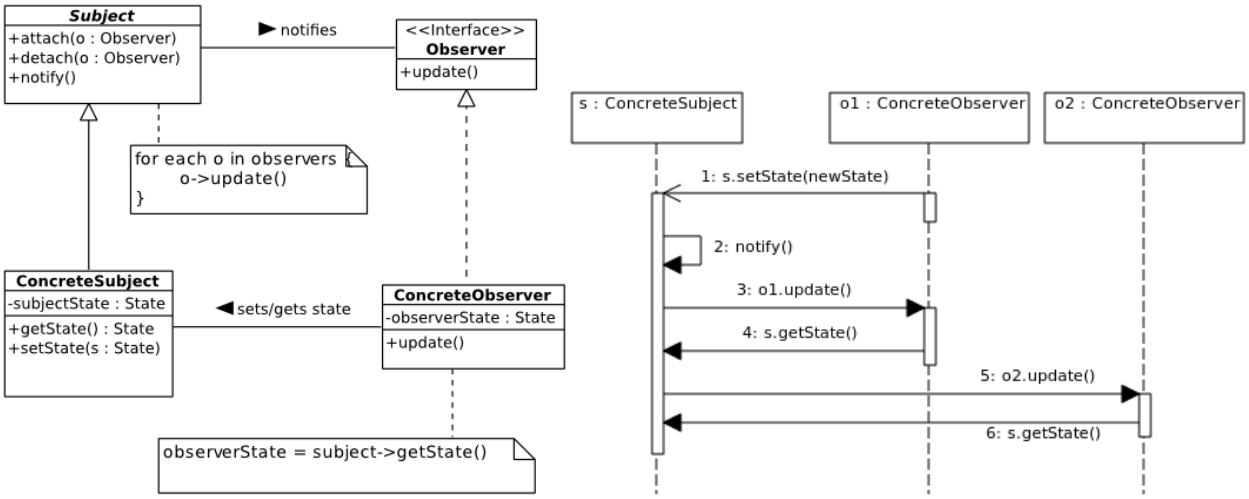
Selon les termes de Gamma et al., il s'agit d'un patron avec un but *comportemental* et une portée d'*objet*. Selon Buschamnn et al., le patron Stratégie est un *patron de conception*.

Contexte et Problème Le patron de conception Stratégie peut être appliqué lorsque [GHJV94] :

- Plusieurs classes liées ne diffèrent que par leur comportement. Les stratégies fournissent un moyen de configurer une classe avec un de ces comportements.
- Plusieurs variantes d'un algorithme sont nécessaires.
- Un algorithme utilise des données dont les clients ne devraient pas avoir connaissance. Le patron Stratégie peut être utilisé pour éviter d'exposer des structures de données complexes et spécifiques à un algorithme.
- Une classe définit plusieurs comportements et il apparaît plusieurs conditions à leurs opérations. Ces branches conditionnelles peuvent être déplacées dans leur propre classe Stratégie.

Conséquences Selon Gamma et al. [GHJV94] :

- *Les familles d'algorithmes liés.* Les hiérarchies des classes de Stratégie définissent une famille d'algorithmes ou de comportements pour la réutilisation.
- *Une alternative aux sous-classements.* L'héritage offre une autre façon de supporter une variété d'algorithmes ou de comportements.
- *Les stratégies éliminent les instructions de condition.*
- *Un choix d'implémentation.* Les stratégies peuvent fournir différentes implémentations d'un même comportement.



(a) Aspect statique (b) Aspect dynamique

FIGURE 2.2 – Le patron Observateur tel que défini par Gamma et al. [GHJV94]

- *Les clients doivent être conscients des différentes Stratégies.* Le patron a un point négatif potentiel en quoi le client doit comprendre comment les stratégies diffèrent avant d'en choisir une appropriée.
- *La communication entre la Stratégie et le Contexte.* L'interface `Strategy` est partagée par toutes les classes `ConcreteStrategy` en fonction de si les algorithmes qu'elles implémentent sont triviaux ou complexes.
- *Augmentation du nombre d'objets.* Les stratégies augmentent le nombre d'objets dans une application.

2.3.2 Patron de conception Observateur

Gamma et al. définissent le patron de conception *Observer* (aussi connu sous les noms de *Dependents* et *Publish-Subscribe* [BMR+96, GHJV94]) comme une dépendance un-à-plusieurs entre des objets tels que lorsqu'un objet change d'état, toutes ses dépendances sont notifiées et mises à jour automatiquement [GHJV94]. L'utilisateur ne veut généralement pas coupler fortement ses classes car cela réduit leur réutilisation. L'utilisation d'un tel patron est un choix d'architecture. La Figure 2.2a illustre la spécification statique de ce patron. Alors que tous les "observateurs" implémentent une interface *Observer* et sa méthode de mise à jour, tous les éléments "observables" (les "sujets") implémentent une même classe qui permet de se lier à un ensemble d'Observateurs pour les notifier d'un changement d'état. La Figure 2.2b présente le scénario dans lequel deux observateurs reçoivent une notification d'un sujet qui vient d'être modifié. Il est supposé que ces observateurs sont préalablement enregistrés auprès du sujet.
Selon les termes de Gamma et al., il s'agit d'un patron avec un but *comportemental* et une portée d'*objet*. Selon Buschamnn et al., le patron Observateur est un *patron de conception*.

Les éléments essentiels de ce patron de conception sont [BMR+96] :

Contexte Son utilisation se présente dans une situation où des données changent à un endroit, mais dont d'autres composants dépendent de ces données.

Problème La solution doit équilibrer les forces suivantes :

- Un ou plusieurs composants doivent être notifiés au sujet de changements d'état dans un composant particulier.

- Le nombre et l'identité des composants dépendants ne sont pas connus à priori ou peuvent éventuellement changer au cours du temps.
- Une demande explicite des dépendants pour une nouvelle information n'est pas possible.
- Les informations du Sujet et de ses dépendants ne devraient pas être étroitement couplées lors de l'introduction du mécanisme de propagation des changements.

Conséquences Plusieurs bénéfices et désavantages sont à mettre en évidence [GHJV94].

- *Couplage abstrait entre le sujet et l'observateur.* Tout sujet sait qu'il dispose d'une liste d'observateurs, chacun se conformant à une simple interface de la classe abstraite `Observer`. Le sujet ne connaît pas la classe concrète des observateurs quels qu'ils soient. Le couplage est donc abstrait et minimal entre les sujets et les observateurs.
- *Support pour la diffusion de communications.* Contrairement à une requête ordinaire, la notification qu'un sujet envoie n'a pas besoin de spécifier son récepteur. La notification est diffusée automatiquement à tous les objets intéressés qui lui ont souscrit. Cela permet la liberté d'ajouter et de supprimer des observateurs à tout moment.
- *Mises à jour non-désirées.* En raison du fait que les observateurs n'ont pas de connaissance de la présence des autres, ils peuvent être aveugles au coût d'une modification du sujet.

2.4 Modèle-Vue-Contrôleur

Le but de cette section est de présenter l'architecture Modèle-Vue-Contrôleur (MVC) et ses variantes. En présentant l'histoire de MVC en premier lieu, nous désirons mettre en exergue la pensée qui sous-tend le besoin de patrons architecturaux dans le contexte des programmes ayant des interfaces utilisateurs flexibles.

2.4.1 Histoire

Dans Smalltalk-76, le précurseur de Smalltalk-80, l'idée était de laisser les objets représenter de l'information d'intérêt à l'utilisateur et également de savoir comment présenter cette information à l'écran pour ensuite laisser l'utilisateur la modifier. Ce mécanisme très puissant est la base des interfaces orientées objets si populaires aujourd'hui. Ce concept s'est montré inadéquat lorsque Reenskaug voulu utiliser Smalltalk-76 pour créer un système de contrôle de production de construction navale. L'information représentée dans le système était le calendrier de production accompagné de ses activités et de ses ressources et l'utilisateur voulait le voir et le manipuler de différentes façons : comme un réseau d'activités, comme un graphique montrant chaque activité sous la forme d'un diagramme de Gantt, et comme une forme d'affaire présentant les activités comme du texte éditable. Une conséquence naturelle fût de diviser l'objet en un objet représentant l'information, un objet présentant cette information et un objet capturant les entrées utilisateurs. Cela permettrait d'avoir différentes présentations et des facilités d'entrée utilisateur pour le même objet [RWL95].

Cette séparation des rôles deviendra plus tard le patron architectural *Model-View-Controller* (MVC). Ce patron est "inventé" au *Xerox Palo Alto Research Center* (*Xerox PARC*) par Reenskaug entre l'été 1978 et l'été 1979. Initialement, Reenskaug travaille sur la métaphore Chose-Modèle-Vue-Éditeur (*Thing-Model-View-Editor*) qu'il développe dans son rapport éponyme [Ree79b]. Effectivement, la pensée initiale de Reenskaug ne se tourne pas directement sur l'architecture MVC. Selon son travail :

- La **Chose** est quelque chose qui intéresse l'utilisateur.
- Le **Modèle** est une représentation active d'une abstraction sous forme de données dans un programme.

- La **Vue** est capable d'afficher une ou plusieurs représentations graphiques du Modèle à l'écran.
- L'**Éditeur** est une interface entre un utilisateur et une ou plusieurs vues.

Au terme de nombreuses discussions, il statue dans son article [Ree79a] sur le terme Modèle-Vue-Contrôleur où le **Modèle** représente la connaissance au travers d'un objet simple ou d'une structure d'objets. La **Vue** est une représentation (visuelle ou non) de son modèle. Le nouveau terme **Contrôleur**, lui, décrit le lien entre un utilisateur et le programme. L'**Éditeur** est dès à présent considéré comme un contrôleur spécial qui permet à l'utilisateur de modifier l'information qui est présente dans la vue.

Goldberg [GR83] améliore la conception de Reenskaug dans son implémentation pour ensuite l'intégrer à Objectworks-Smalltalk [RWL95].

Deacon [Dea09] met en évidence que le terme Modèle tel qu'il est défini avec Smalltalk est ambigü. Il propose, selon lui, un meilleur accronyme pour l'architecture MVC : $M_d M_a VC$ où M_d représente le modèle du domaine (*domain model*) et M_a est le modèle d'application (*application model*). Cette distinction est à garder à l'esprit car certains auteurs s'y réfèrent pour davantage de précision.

- Le **modèle du domaine** consiste en objets qui représentent et supportent l'essence du problème.
- Le **modèle d'application** est l'objet qui a connaissance des vues et qui permet aux vues d'obtenir l'information et les notifications.

2.4.2 Implémentation originale

Lorsque Reenskaug passe de sa construction mentale de Chose-Modèle-Vue-Éditeur à sa conception de Modèle-Vue-Contrôleur, MVC se présente comme une méthode et une pensée. Il est ensuite nécessaire de considérer cette pensée au regard du paradigme orienté objet. Reenskaug est le premier à imaginer une implémentation en 1978. Il soulève un aspect, travaillé par Althoff, qui sera parfois revu par les versions postérieures de MVC [Ree10] :

> "*U*n aspect important du MVC original était que son Contrôleur était responsable de la création et de la coordination de ses vues subordonnées. "

Dans les prochaines implémentations de Reenskaug, une vue accepte et gère les entrées utilisateurs qui lui sont pertinentes. La Figure 2.3 illustre l'architecture MVC telle qu'imaginée par Reenskaug. Cette architecture présente cinq éléments : l'utilisateur, le contrôleur, la vue, le modèle et l'outil (*Tool*). Alors que le composant outil est composé d'un contrôleur lié à plusieurs vues (où chaque vue ne dispose que d'un seul contrôleur), cet outil est en liaison avec plusieurs modèles (et chaque modèle peut être lié à plusieurs outils). En outre, l'outil est le point de connexion entre l'utilisateur et le modèle. Cette Figure fait la distinction du terme Modèle qui peut porter à confusion s'il fait référence au programme ou à l'utilisateur. Le composant Modèle correspond à la version "informatisée" du modèle mental qu'a l'utilisateur du problème auquel il est confronté avec l'outil.

Une implémentation significative de MVC est décrite par Krasner et Pope [KP88]. Ils décrivent un standard pour le cycle d'interaction de MVC comme illustré en Figure 2.4.

> "*L*e cycle d'interaction standard dans la métaphore Modèle-Vue-Contrôleur consiste en un utilisateur qui donne une certaine entrée au contrôleur actif qui notifie le modèle pour qu'il se modifie lui-même en conséquence. Le modèle effectue les opérations prescrites, change potentiellement son état et annonce à ses dépendances (vues et contrôleurs) qu'il a changé, en leur donnant éventuellement la nature des changements. Les vues

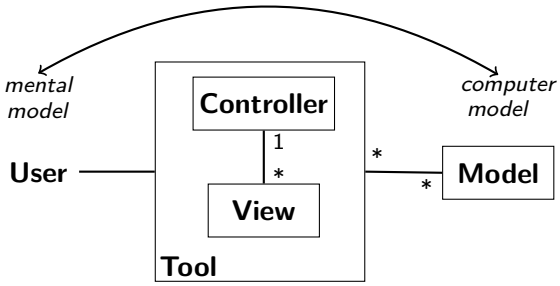


FIGURE 2.3 – La conception mentale originale de MVC par Reenskaug [Ree10]

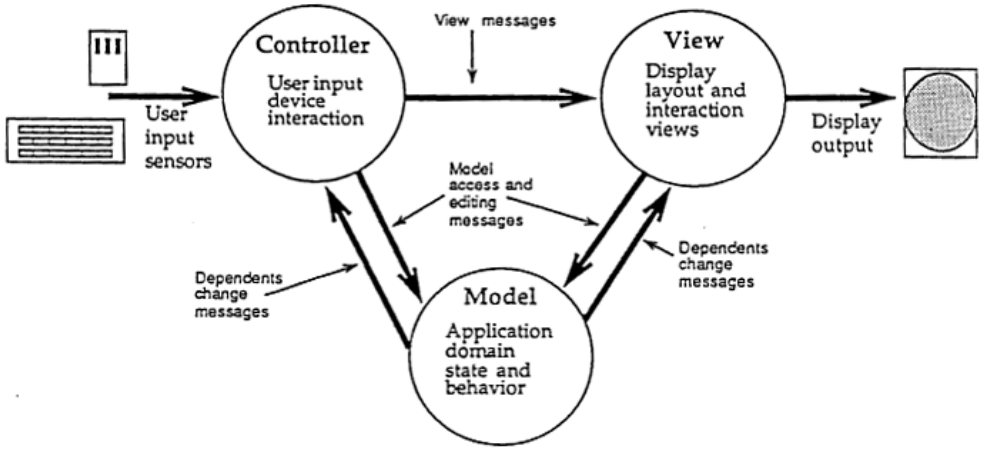


FIGURE 2.4 – Le cycle d'interaction standard dans la métaphore Modèle-Vue-Contrôleur [KP88]

peuvent ensuite se renseigner auprès du modèle au sujet de son nouvel état et mettre à jour leur affichage si nécessaire. Les contrôleurs peuvent changer leurs méthodes d'interaction en fonction du nouvel état du modèle. ”

2.4.3 MVC comme patron architectural

Les éléments constitutifs relatifs aux patrons tels que décrit en Section 2.1.6 se retrouvent également dans le patron architectural qu'est MVC. Alors que l'élément de nom est trivial, les éléments de contexte, de problème, de solution et de conséquences méritent d'être détaillés.

Le contexte

MVC s'inscrit dans le paradigme orienté objet. Le contexte dans lequel MVC est utile est un programme nécessitant une ou plusieurs interfaces utilisateurs flexibles [BMR+96].

Le problème

Les interfaces utilisateurs sont généralement sujettes aux changements. Lorsque les fonctionnalités d'un programme sont étendues, il est souvent nécessaire de modifier les interfaces qui leurs sont liées. Un client peut faire une demande spécifique de modification d'interface ou un programme peut nécessiter d'être porté vers une autre plate-forme avec un *Look and Feel* différent. Même la mise à niveau vers une nouvelle version d'un système de fenêtrage peut impliquer des changements de code. Dans les systèmes d'une grande longévité, l'interface utilisateur des programmes d'une grande longévité est un élément sujet à de fréquentes modifications [BMR+96].

Les exigences des différents utilisateurs sont parfois sujettes à conflit quant aux interfaces utilisateurs. En conséquence, le support de plusieurs paradigmes d'interface utilisateur doit être incorporé facilement. Construire un système avec la flexibilité requise est coûteux et enclin aux erreurs si l'interface utilisateur est intimement liée au coeur de fonctionnalités. Cela peut résulter par le besoin de développer et de maintenir plusieurs systèmes logiciels considérablement différents, un pour chaque implémentation d'interface utilisateur. Les modifications qui en découlent sont réparties sur plusieurs sous-programmes. Les points forts suivants influencent la solution [BMR+96] :

- La même information est présentée différemment dans des fenêtres différentes (par exemple, dans un graphique en nuage de points ou en camembert).
- L'affichage et le comportement de l'application doit refléter immédiatement les manipulations de données.
- Les modifications de l'interface utilisateur doivent être simples et parfois même être possibles à l'exécution.
- Le support de différents standards de *Look and Feel* ou le portage des interfaces utilisateurs ne doit pas affecter le code du coeur de l'application.

Selon Krasner et Pope [KP88], l'objectif du patron MVC est de satisfaire deux buts :

1. Créer l'ensemble spécial de composants systèmes nécessaires pour supporter un processus de développement hautement interactif.
2. Fournir un ensemble général de composants systèmes pour permettre aux programmeurs de créer facilement des "interfaces graphiques modulaires" (*portable interactive graphical applications*, selon le texte original).

La solution

MVC divise une application interactive en trois domaines : le traitement (*processing*), les entrées et les sorties.

Le composant **Modèle** contient le cœur des fonctionnalités de l'application. Il encapsule les données appropriées et exporte les méthodes qui effectuent les traitements spécifiques à l'application. Les contrôleurs appellent ces procédures pour le compte de l'utilisateur. Le modèle fournit également des fonctions pour accéder à ses données qui sont utilisées par les vues pour acquérir les données à afficher. Le **mécanisme de propagation des changements** maintient un registre des composants à l'intérieur du modèle. Toutes les vues ainsi que les contrôleurs sélectionnés enregistrent leur besoin d'être informés au sujet des modifications de données. Ces modifications d'état du modèle déclenchent le mécanisme de propagation des changements. Ce mécanisme fait uniquement le lien entre le modèle et les vues et les contrôleurs. Il est décrit par le patron de conception Observateur détaillé en Section 2.3.2.

Les composants de **Vue** présentent l'information à l'utilisateur. Les différentes vues exposent l'information du modèle de différentes façons. Chaque vue définit une procédure de mise à jour qui est activée par le mécanisme de propagation des changements. Lorsque la procédure de mise à jour est appelée, les vues récupèrent la valeur courante des données à partir du modèle et les affichent à l'écran. Lors de l'initialisation, toutes les vues sont associées au modèle et sont enregistrées dans le mécanisme de propagation des changements. Chaque contrôleur crée une vue appropriée. Il existe une relation un-à-un entre les vues et les contrôleurs. Les vues offrent souvent des fonctionnalités qui permettent aux contrôleurs de manipuler l'affichage. Certaines opérations déclenchées par les

utilisateurs peuvent ne pas affecter le modèle (par exemple, le *scrolling*).

Les composants **Contrôleurs** acceptent les entrées utilisateurs telles que les évènements. La manière d'offrir ces évènements à un contrôleur dépend de la plate-forme de l'interface utilisateur. Les évènements sont traduits en requêtes pour le modèle ou les vues associées. Si le comportement du contrôleur dépend de l'état du modèle, le contrôleur s'enregistre lui-même avec le mécanisme de propagation des changements et implémente une procédure de mise à jour. Par exemple, ce mécanisme entre le Contrôleur et le Modèle est nécessaire lorsqu'un changement du modèle active ou désactive une entrée du menu.

Les conséquences

Plusieurs bénéfices sont à considérer lors de la mise en application du patron architectural MVC [BMR+96] :

De multiples vues pour un même modèle MVC sépare strictement le modèle des composants de l'interface utilisateur. Par conséquent, de multiples vues peuvent être implémentées et utilisées avec un seul modèle. À l'exécution, de multiples vues peuvent être ouvertes en même temps.

Vues synchronisées Le mécanisme de propagation des changements du modèle assure que tous les observateurs sont notifiés des modifications des données de l'application au moment approprié. Cela synchronise toutes les vues et les contrôleurs dépendants.

Des vues et des contrôleurs *pluggable* La séparation conceptuelle de MVC permet d'échanger les objets de vue et de contrôleur d'un modèle. Les objets d'interface utilisateur peuvent également être substitués à l'exécution.

Échangeabilité du *Look and Feel* Parce que le modèle est indépendant de tout le code de l'interface utilisateur, un "portage" d'une application MVC vers une nouvelle plate-forme n'affecte pas le cœur fonctionnel de l'application. Pour chaque plate-forme, seules sont nécessaires des implémentations des vues et des contrôleurs.

Potentiel du *framework* Il est possible de reposer un *framework* d'application sur ce patron architectural comme l'a prouvé SmallTalk.

Néanmoins, des handicaps sont également à déplorer [BMR+96] :

Complexité plus élevée Suivre la structure MVC de façon stricte n'est pas toujours la meilleure façon de construire une application interactive. Gamma soutient que l'utilisation de composants séparés pour les menus et les éléments de texte simple augmentent la complexité sans pour autant gagner en flexibilité.

Nombre de mises à jour parfois excessif Si une simple action d'un utilisateur résulte en plusieurs mises à jour, le modèle devrait éviter des notifications de modification inutiles. Il se peut que toutes les vues ne soient pas intéressées par chaque modification propagée par le modèle. Par exemple, une vue avec une fenêtre icônisée peut ne pas être nécessairement mise à jour jusqu'à ce que la fenêtre reprenne son état normal.

Lien intime entre la vue et le contrôleur Le contrôleur et la vue sont séparés mais sont des composants intimement liés, ce qui empêche leur réutilisation individuelle. Il est peu probable qu'une vue soit utilisée sans son contrôleur, ou vice-versa, avec l'exception des vues en lecture seule qui partagent un contrôleur qui ignore toute entrée utilisateur.

Couplage étroit entre les vues et les contrôleurs à un modèle Les composants de vue et de contrôleur font tous deux des accès directs au modèle. Cela implique que les changements d'interface du modèle sont susceptibles d'impliquer des changements dans le code de la vue et du

contrôleur. Ce problème est mis en exergue si le système utilise une multitude de vues et de contrôleurs. Par exemple, l'application du patron de conception *Command Processor* [GHJV94] peut résoudre ce problème.

L'inefficacité de l'accès des données dans les vues En dépendant de l'interface du modèle, une vue peut nécessiter de faire de multiples appels pour obtenir tous ses affichages de données. Des demandes de données inchangées au modèle affaiblissent la performance si les mises à jour sont fréquentes.

Changement inévitable des vues et des contrôleurs lors d'un portage Toutes les dépendances sur la plate-forme de l'interface utilisateur sont encapsulées à l'intérieur de la vue et du contrôleur. Cependant, les deux composants contiennent également du code qui est indépendant d'une plate-forme spécifique. Le portage d'un système MVC requière donc la séparation du code dépendant de la plate-forme avant de le réécrire.

2.5 Variantes de MVC

Au fil du temps, différentes implémentations et différentes extensions de l'architecture MVC ont été développées. Elles permettent parfois de résoudre des inconvénients de l'architecture initiale dans des contextes spécifiques et elles mettent parfois simultanément en danger certains avantages de MVC. Nous citons ci-dessous les variantes les plus significatives.

2.5.1 MVC avec les patrons Observateur et Stratégie

Le mécanisme de propagation des changements soulève un problème général qu'est le lien entre le modèle et les vues. Cette conception générale est décrite ici par le patron de conception Observateur [GHJV94]. En effet, la plupart des patrons de conception pour l'architecture logicielle soulèvent des problèmes qui peuvent être résolus par de "plus petits" patrons [BMR+96, p. 16]. Gamma et al. déclarent dans [GHJV94] :

> "*L*e premier exemple, et peut-être le mieux connu, d'application du patron de conception Observateur apparaît dans le MVC de SmallTalk, le *framework* d'interface utilisateur de l'environnement SmallTalk. La classe de Modèle de MVC joue le rôle du Sujet alors que la Vue est la classe de base pour l'observateur. "

Il s'avère donc que le patron de conception Observateur (décrit en Section 2.3.2) est tout indiqué pour réaliser la communication entre la vue et le modèle du patron architectural MVC. Une implémentation connue de cette version de MVC est *Interviews* [LVC89]. Contrairement à la description apportée par Buschmann et al. [BMR+96], le *Gang of Four* ne tient pas compte du contrôleur comme Observateur. Il en va de même pour l'implémentation Smalltalk. C'est cette version dont nous tiendrons compte pour la suite de ce document.

Les mêmes auteurs considèrent que l'architecture MVC peut également être composée avec un autre patron de conception : le patron Stratégie présenté en Section 2.3.1. Ce patron de conception n'est utilisé avec l'architecture MVC que dans des cas spécifiques. Il permet la gestion de différents comportements au niveau du contrôleur pour des vues identiques.

En conclusion, le lien entre le modèle et la vue est simplifié par l'utilisation d'un patron Observateur alors que le patron Stratégie rend la gestion des interactions entre le contrôleur et la vue plus simple. Il est possible de représenter l'architecture MVC augmentée de ces deux patrons à l'aide du diagramme de classes UML présenté sur la Figure 2.5.

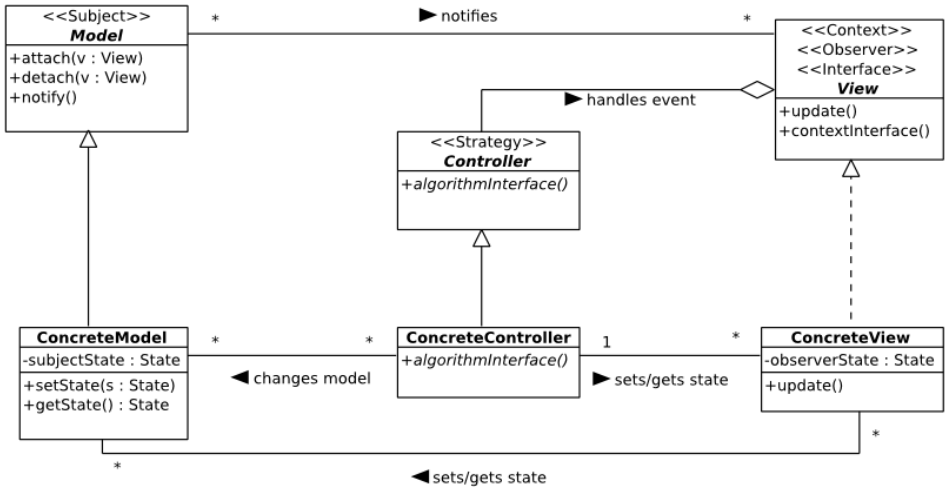


FIGURE 2.5 – L'architecture MVC augmentée des patrons de conception Observateur et Stratégie

2.5.2 *Model-Delegate*

Dans plusieurs plate-formes d'interface utilisateur (GUI), l'affichage des fenêtres et la gestion des évènements sont intimement liés. Par exemple, le système de fenêtrage X11 [4] signale les évènements relatifs à une fenêtre. Il est possible de combiner les responsabilités de la vue et du contrôleur de MVC dans un simple composant en sacrifiant l'échangeabilité des contrôleurs [BMR+96].

Model-Delegate est une variante de MVC aussi appelée *UI-Model* (par Swartz [Swa04], par exemple) ou *Document-View* [BMR+96]. Les parties relatives aux vues et aux contrôleurs sont agrégées en un composant nommé *User Interface Delegate*. Le rôle distinct de chaque composant doit donc être revu et spécifié selon le paradigme orienté objet :

- Les classes de **contrôleur/vue (UI pour *User Interface*)** présentent l'information et gèrent les interactions avec l'utilisateur.
- Les classes du **Modèle** détiennent les structures de données spécifiques au programme. Ces classes n'ont aucune connaissance du composant *User Interface Deletage*.

Par exemple, l'architecture de Java Swing est basée sur la conception MVC. Le premier prototype suivait la séparation traditionnelle de MVC. Néanmoins, cette conception pose nombre de problèmes et Swing réduit finalement la vue et le contrôleur dans un *UI Object* (*user-interface*) comme l'explique Fowler :

> "*N*ous avons rapidement découvert que cette séparation ne fonctionnait pas bien en termes pratiques car la vue et le contrôleur sont des composants qui requièrent un couplage étroit (par exemple, il était très difficile d'écrire un contrôleur générique qui ne connaissait pas les spécificités de la vue). " [Sun]

La communication entre le Modèle et le UI est définie selon Swartz telle que :

> "*L*e UI et le modèle doivent communiquer et la forme principale de communication entre les éléments du programme sont les méthodes d'appel. Une bonne méthode commune qui est employée est de créer un objet du modèle qui peut être utilisé par le UI pour appeler les méthodes du modèle. Chaque fois que l'utilisateur fait quelque chose dans l'UI, les *listeners* utiliseront cet objet pour appeler les méthodes appropriées dans le modèle. " [Swa04]

4. `http://www.x.org/`

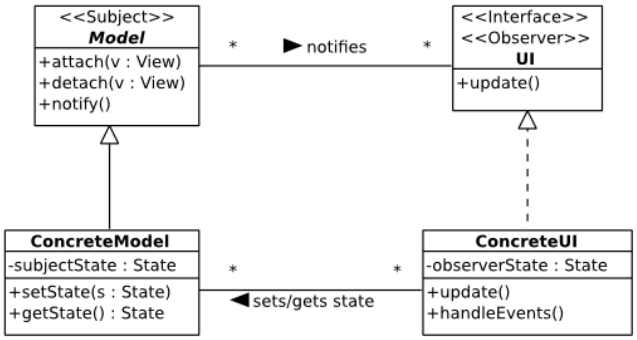


FIGURE 2.6 – L'architecture *Model-Delegate* augmentée du patron de conception Observateur

Afin de mettre le composant UI à jour, il existe donc plusieurs méthodes. Dans un cas simple, l'affichage des valeurs de retour des méthodes du Modèle suffit. Les vues peuvent cependant être plus complexes, auquel cas une solution possible est d'avoir dans le composant UI des *listeners* constamment connectés au Modèle. Lorsque le Modèle change, chacun de ces *listeners* est notifié des changements d'états. Il s'agit là d'un procédé similaire au patron de conception Observateur [GHJV94]. Comme dans MVC, le faible couplage du modèle et de la vue rend possible de multiples vues simultanément synchronisées pour un même modèle [BMR+96].

La Figure 2.6 représente la spécification de l'architecture *Model-Delegate* au travers d'un diagramme de classes UML.

2.5.3 Modèle-Vue-Présentateur

Le patron architectural *Model-View-Presenter* (MVP) est inventé par Potel, employé chez IBM, en 1996. À l'instar de l'architecture *Model-Delegate*, il remet en question la séparation des composants de MVC [Pot96] :

> "La première étape dans ce traitement a été de formaliser la séparation entre le modèle et la Vue-Contrôleur, auquel nous nous référons en tant que "Presentation". Cela représente la décomposition d'un problème de programmation en deux concepts fondamentaux : la gestion des données et l'interface utilisateur. [...] Pour la gestion des données, la question que le programmeur se pose est "Comment puis-je gérer mes données ?". [...] Pour l'interface utilisateur, la question du programmeur est "Comment l'interface utilisateur interagit-elle avec mes données ?". "

La Figure 2.7 illustre la conception mentale qu'a Potel de cette nouvelle séparation. La séparation des composants Modèle/Vue/Présentateur conserve l'idée initiale du Modèle (en terme de modèle du domaine) et de la Vue. Néanmoins, le composant Présentateur se pose comme le lien entre les deux autres et a pour objectif de répondre à la question "Comment assembler tous les composants ensemble ?" en plus de jouer le rôle de Contrôleur. Son rôle d'intermédiaire lui permet de "présenter" les données du modèle à la vue en appliquant des modifications si nécessaire. Outre ces trois éléments fondamentaux, Potel décrit trois autres éléments intermédiaires.

1. L'*Interactor* gère les évènements relatifs aux interactions de l'utilisateur et aux *gestures* [5].
2. La Commande (*Command*) spécifie les opérations qui modifient les données. Néanmoins, ce composant n'a pas de connaissance sur la manière dont sont effectuées ces opérations. Il s'agit

5. Il est entendu par *gestures* les séquences d'évènements relatifs à une tâche particulière.

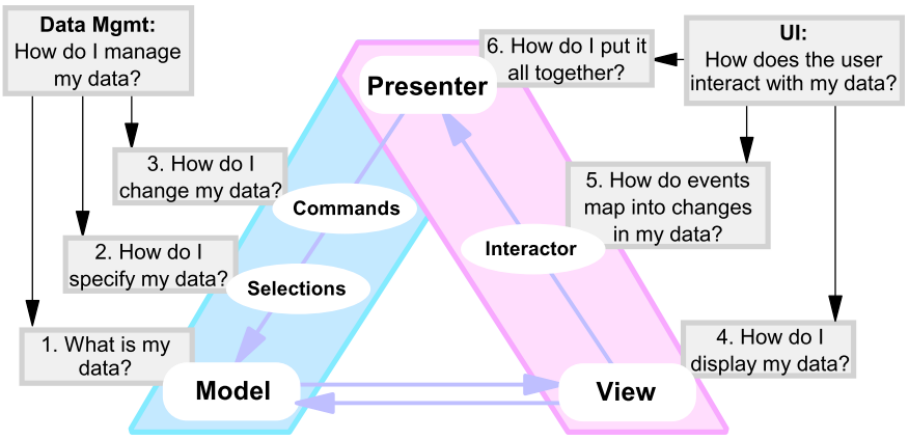


FIGURE 2.7 – Représentation mentale de l'architecture MVP [Pot96]

simplement d'un espace réservé pour les objets qui encapsulent les opérations, appelés "objets de commande" [6] [Law08].

3. Les Sélections (*Selections*) désignent quels ensembles de données doivent être modifiés par les objets de commande.

Alors que l'*Interactor* se situe au niveau des composants Vue et Présentateur, les Commandes et les Sélections sont intégrées au Modèle. L'implémentation de Potel, plus connue sous le nom de *Taligent's MVP*, utilise le patron de conception Observateur afin de réaliser les notifications du Modèle à ses Vues [Law08].

L'architecture MVP est sujette à plusieurs adaptations selon les auteurs qui la spécifient. En effet, bien que l'article original de Potel fournisse beaucoup d'information, il persiste nombre de contradictions et d'incertitudes au sujet des questions d'implémentation, le rôle de chaque composant et du flux de contrôle [Law08]. Fowler établit que Bower et McGlashan [AB00] ont une conception différente [7] du rôle du Présentateur :

> "*U*ne des variations dans la pensée au sujet de MVP est le degré avec lequel le Présentateur contrôle les objets graphiques dans la Vue. D'un côté, il existe le cas où toute la vue logique est laissée dans la vue et où le présentateur n'est pas impliqué dans l'affichage des éléments du modèle. Ce style est celui adopté par Potel. La direction prise par Bower et McGlashan était ce que nous appelons *Supervising Controller*, où la vue gère une bonne partie de la vue logique qui peut être décrite déclarativement et le présentateur vient ensuite pour gérer les cas les plus complexes. Il est également possible de considérer le cas où le présentateur fait toutes les manipulations des objets graphiques. Cette méthode, que nous appelons "Vue Passive", n'est pas dans les descriptions originales de MVP mais s'est développée lorsque les gens se sont intéressés aux problèmes de testabilité. "[Fow06]

La Figure 2.8 illustre l'architecture MVP selon la pensée de Potel à l'aide d'un diagramme de classes. Une fois les éléments intermédiaires agrégés à leur composant principal, ce patron peut être représenté de façon plus accessible comme une variante de MVC (Figure 2.9). La version présentée par la Figure 2.9 reste proche de l'architecture originale de MVC si ce n'est que le composant de Présentation est un Contrôleur qui permet la conversion de données du modèle afin de les présenter à la Vue dans un état approprié [MSD]. C'est cette version simplifiée, proche de MVC, qui sera considérée tout au long de ce document.

6. Ces objets utilisent le patron de conception *Command* [GHJV94].
7. nommée Dolphin's MVP

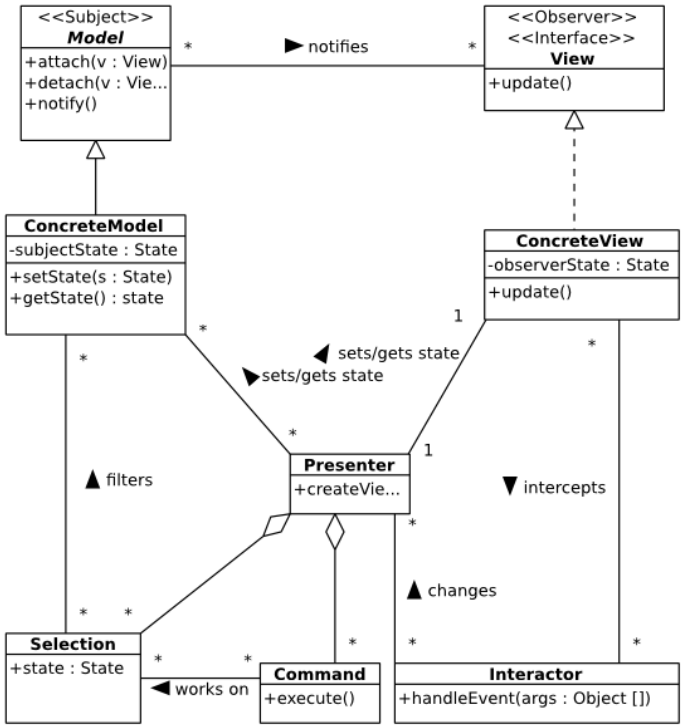


FIGURE 2.8 – Représentation théorique de l'architecture MVP

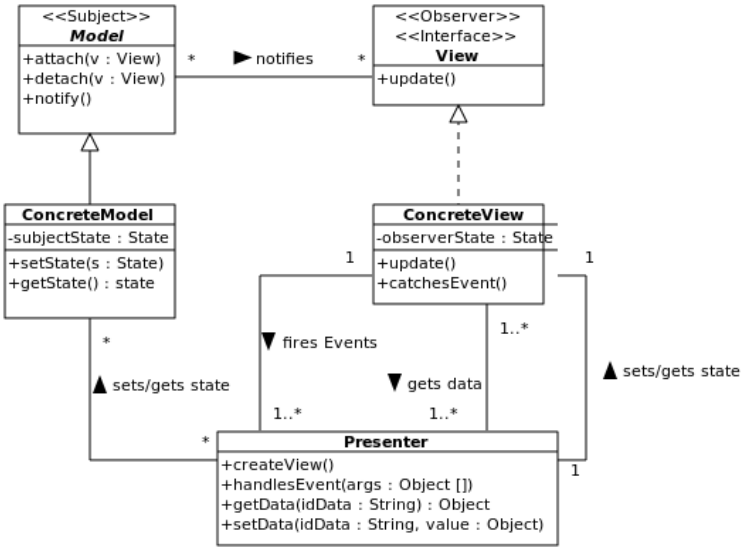


FIGURE 2.9 – Représentation pratique de l'architecture MVP

2.5.4 *Presentation Model*

Alors que les patrons présentés jusqu'ici ont nombre d'avantages et d'inconvénients, aucun ne gère la persistance de la Vue. Une telle persistance permettrait de conserver les éléments de la vue dans l'état où ils se trouvent à un moment donné.

> "*A*ussi puissants qu'ils sont, MVC et MVP ont leurs problèmes. Un de ces problèmes est la persistance de l'état de la Vue. Par exemple, si le Modèle, comme objet du domaine, ne connaît rien au sujet de l'interface utilisateur (UI), et que la Vue n'implémente aucune logique métier, alors où serait stocké l'état des éléments de la vue tel qu'un élément sélectionné dans une liste ? " [She09]

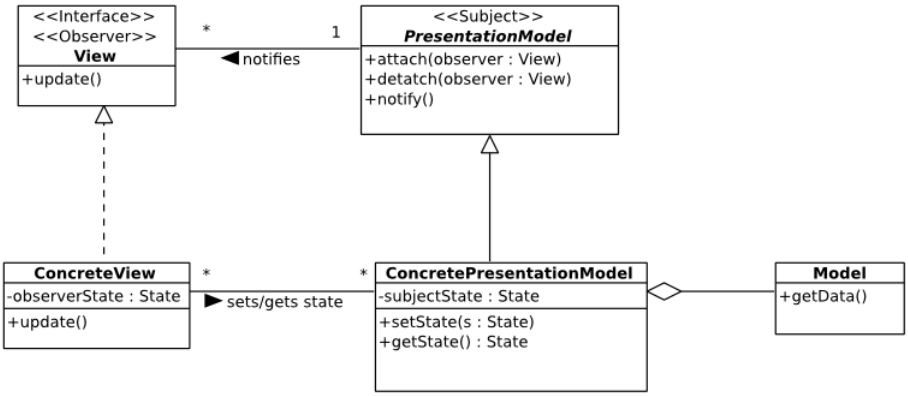


FIGURE 2.10 – L'architecture *Presentation Model* [She09]

Fowler [Fow04] propose une telle solution qui introduit une classe (nommée *ApplicationModel* ou *Presentation Model*) entre le Modèle et la Vue afin de conserver l'état de la Vue. Cette nouvelle classe devient donc le nouveau Modèle de la Vue initiale et devient la nouvelle Vue du Modèle initial. L'*ApplicationModel* sait comment mettre à jour l'UI mais ne référence aucun élément de l'UI directement. La classe *PresentationModel* enrichit les données du modèle avec de l'information telle que l'état de la vue qui est synchronisée avec la vue en utilisant le patron *Publish-Subscribe* [She09]. Cette nouvelle classe minimise les prises de décision dans la vue [Fow04]. Cette variante de MVC est représentée sur la Figure 2.10.

2.5.5 Autres variantes

Selon l'architecte Alexander,

> "Chaque patron dépend des patrons plus petits qu'il contient et du patron plus grand qui le contient. " [Ale79]

La littérature mentionne un grand nombre d'autres variantes de l'architecture MVC. Comme énoncé en Section 2.5.1, les patrons soulèvent des problèmes qui peuvent être résolus par d'autres patrons. Cette règle implique qu'en fonction du contexte, il est possible d'adapter l'architecture MVC avec d'autres patrons pour en obtenir une nouvelle variante ([BMR+96, p. 396] en propose plusieurs). Par conséquent, le nombre de variantes possibles à MVC sont trop nombreuses pour pouvoir toutes les citer mais en voici quelque-unes.

Gossman propose le patron *Model/View/ViewModel* où le Modèle est identique à l'architecture MVC. La Vue consiste en des éléments graphiques et aux interactions avec l'utilisateur. La Vue joue donc le rôle de la Vue et du Contrôleur de MVC. L'élément *ViewModel* qui doit être compris comme "Modèle de la Vue" et fournit une spécialisation du Modèle. Cette spécialisation convertit les types de données du Modèle en type de données de la Vue [Gos05]. Cette architecture est davantage décrite en détail par Smith [Smi09]. À l'instar du patron *Presentation Model*, la persistance de la vue est également possible.

Une idée similaire est présentée par Rammerstorfer et Mössenböck [RM03]. Il s'agit d'une couche de *data mapping* qui consiste en un ensemble d'éléments de *mapping* qui peuvent être connectés ensemble pour réaliser une certaine transformation. Le but de ce *mapping* est de concilier un modèle et ses vues si leurs structures ne sont pas identiques. Ces *mappings* peuvent être composés hiérarchiquement en utilisant les patrons de conception *Decorator* et *Composite* [RM03]. Cette architecture a donc un but

commun avec le patron *Model/View/ViewModel* si ce n'est qu'il conserve le Contrôleur de l'architecture MVC.

2.5.6 Applications

Ces patrons architecturaux ont largement été utilisés depuis de nombreuses années. MVC donne lieu à des programmes (dont des applications Web) tels que SpringMVC [Spr10], PureMVC pour Adobe Flex [Hal08], Cocoa [App10] (dont le Contrôleur est augmenté d'un patron de conception Médiateur [GHJV94]) et l'environnement VisualWork de SmallTalk [GR83, LP91]. Par exemple, le *framework* Web Struts2 [The] emploie une version de MVC qui ne dispose que d'un seul contrôleur (cette version se nomme MVC2).

Par exemple, MFC [Kru96] implémente le patron *Model-Delegate*. Il en est de même pour Java Swing [8] [Sun]. Selon Fowler, la plupart des patrons employés sur le Web sont basés sur le principe de ce patron [Fow02].

Le patron architectural plus récent qu'est MVP est implémenté dans le *framework* Google Web ToolKit (GWT) [Ker10, p. 59].

8. Si *Sun Microsystems* (maintenant Oracle) définit MVC dans sa forme originale, Java Swing ne suit pas cette spécification [Sun02]. Le choix de Sun, bien que fondamentalement basé sur les spécifications MVC originales, se distingue par l'agrégation des vues et des contrôleurs.

CHAPITRE 3

Maintenabilité des produits logiciels

> Any fool can write code that a computer can understand. Good programmers write code that humans can understand.
>
> Martin Fowler

ALORS *le pilier central de ce travail est la maintenabilité, il est nécessaire de définir ce terme. En effet, nombre d'auteurs mentionnent la "maintenabilité" sans en préciser la portée. Plusieurs modèles de qualité, décrits en Section 3.1, caractérisent ce terme. La Section 3.2 reprend les concepts énoncés par ces modèles de qualité pour exposer notre utilisation du terme "maintenabilité".*

La Section 3.1 est majoritairement inspirée de Pfleeger et al. [PA09].

Sommaire

3.1 Qualité des produits

Les ingénieurs logiciels se doivent de trouver des moyens d'évaluation pour s'assurer que leurs produits sont d'une qualité et d'une utilité acceptable. En conséquence, une bonne ingénierie des logiciels doit toujours inclure une stratégie pour produire des logiciels de qualité.

> "*L*es produits étant plus concrets que les processus et les ressources étant donc plus facilement mesurables, il n'est pas surprenant que la plupart des travaux sur les méthodes de mesure soient dirigées vers ce domaine. De plus, les clients encouragent l'évaluation de la qualité des logiciels car ils s'intéressent aux caractéristiques du produit final, indifféremment du processus qui l'a produit. " [PJCK97]

Selon Garvin, la qualité est perçue différemment en fonction des personnes qui l'évaluent. La différence de perception varie car l'importance des caractéristiques de qualité dépendent sur la personne qui analyse le logiciel. Il décrit la qualité selon cinq perspectives :

Vue transcendante La qualité est quelque chose qu'il nous est possible de reconnaître mais pas de définir.

Vue de l'utilisateur La qualité en fonction des objectifs de l'utilisateur.

Vue de manufacture La qualité du produit est conforme à ses spécifications.

Vue du produit La qualité est liée aux caractéristiques inhérentes au produit.

Vue basée sur la valeur La qualité du produit dépend de la quantité de clients qui sont prêt à payer pour en disposer.

Plusieurs organismes tentent de définir des standards de qualité comme par exemple l'*IEEE Standard* [IEE91, p. 163]. L'Organisme International pour la Standardisation (ISO) définit la notion de qualité dans son standard ISO-9000 :2000 :

> "*A*ptitude d'un ensemble de caractéristiques intrinsèques d'un produit, d'un système ou d'un processus à satisfaire les besoins ou les attentes formulés, habituellement implicites, ou imposés [. . .] des clients et autres parties intéressées. " [ISO00a]

La notion de qualité est importante dans un projet informatique car il existe non seulement un coût de qualité (réalisation du projet, reprise, . . .) mais également un coût de non-qualité tels que les coûts d'anomalies internes (détection des défauts, réparation, modifications. . .) et les coûts d'anomalies externes (perte de clients, pénalités, assistance technique. . .) [Hab10].

3.1.1 Modèles de qualité de produit

Les processus de développement produisent un certain nombre d'artefacts tels que les exigences, la conception, les composants du code, les cas de test, les scripts de test, les guides d'utilisateur, etc. Dans chaque cas, il est possible d'examiner un produit pour déterminer s'il dispose des attributs désirés. Il existe plusieurs modèles de qualité qui suggèrent des façons de connecter les différents attributs de qualité. Chaque modèle aide à comprendre comment les plusieurs facettes contribuent à un tout. Nous présentons, ci-dessous, les modèles de McCall, de Boehm et ISO/IEC 9120. D'autres modèles existent tels que FURPS et le modèle de Dromey. Les modèles de qualité comprennent des définitions précises des attributs, des relations entre ces attributs et des méthodes d'évaluation ou d'estimation.

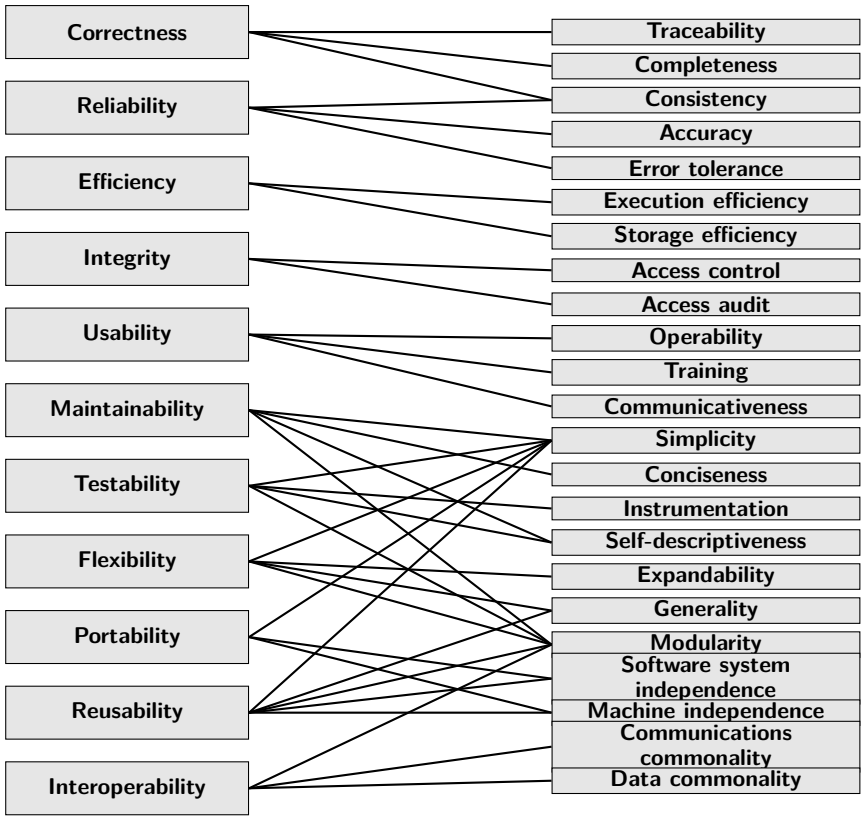


FIGURE 3.1 – Modèle de qualité de McCall [MW77]

Modèle de McCall

McCall est un précurseur dans le domaine et tente de combler le vide entre les utilisateurs et les développeurs en se concentrant sur un nombre de facteurs de qualité qui reflètent les vues des utilisateurs et les priorités des développeurs. Ce modèle est illustré en Figure 3.1 où chaque facteur de qualité externe (dans la rangée de gauche) est lié aux critères de qualité d'un produit (dans la rangée de droite) [MW77]. En outre, le modèle de McCall définit des mesures pour ses critères de qualité.

Modèle de Boehm

Boehm et ses collègues ont construit un des modèles de qualité les plus connus. Il est similaire à celui de McCall dans le sens où il présente une hiérarchie de caractéristiques, chacune d'elles contribuant à la qualité globale du produit [BBK+78, PA09]. Néanmoins, il ne définit aucune mesure mais déclare des "primitives" qui sont en réalité des bases de mesure [Hab10]. Ce modèle (illustré en Figure 3.2) reflète une compréhension de la qualité où le logiciel

- fait ce qui est voulu par l'utilisateur,
- utilise les ressources de l'ordinateur correctement et avec efficience,
- est facile à apprendre et à utiliser pour les utilisateurs,
- est bien conçu, bien codé et est facilement testable et maintenable.

Modèle ISO/IEC 9126

Dans les années 1990, la communauté des ingénieurs du logiciel tente de consolider les plusieurs vues de la qualité en un modèle qui pourrait agir comme un standard mondial. Le standard aurait alors un but de reconnaissance par tous, la qualité de complétude, pourrait être lié à d'autres normes

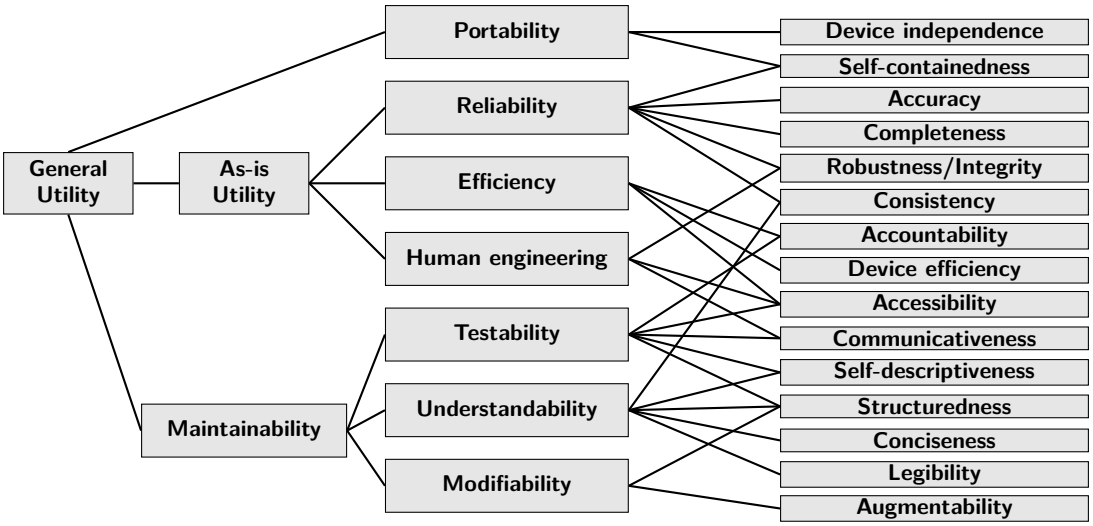


FIGURE 3.2 – Modèle de qualité de Boehm [BBK+78]

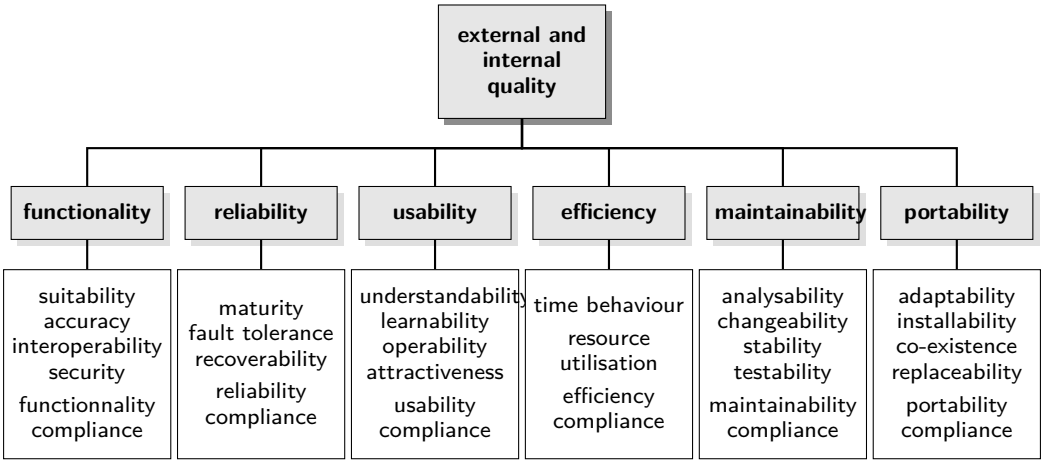


FIGURE 3.3 – Modèle de qualité de ISO/IEC 9126 pour la qualité interne et externe [ISO99, p. 7]

et disposerait du suivi de son évolution [Hab10]. Le résultat est ISO/IEC 9126 [ISO00b] (présenté en Figure 3.3), un modèle hiérarchique avec six attributs majeurs contribuant à la qualité. Une différence avec le modèle de Boehm est que la hiérarchie de ISO est stricte. Le contenu de la norme ISO/IEC 9126 est repris par la norme ISO 25000.

3.1.2 Qualités internes et externes

Selon la norme ISO/IEC 9126, les qualités peuvent être internes, externes et "*in use*". Cette dernière est relative à la perception de qualité de l'utilisateur du produit [ISO99, p. 5], nous n'en tiendrons donc pas compte.

Les **métriques internes** peuvent être appliquées à des produits logiciels non-exécutables (tels qu'une spécification ou un code source) pendant la conception et l'implémentation. Les **métriques externes** utilisent des mesures de produits logiciels dérivées de mesures du comportement du système dont il fait partie, en testant, en exploitant et en observant le logiciel ou le système exécutable [ISO99, p. 14]. ISO propose les mêmes noms pour les caractéristiques internes et externes. Elles se distinguent par les méthodes qui permettent de les calculer.

3.2 Maintenabilité

Notre intérêt, dans ce document, est de s'intéresser à la **maintenabilité interne** des programmes. Selon le standard ISO/IEC 9126 :

> "*L*es métriques de maintenabilité interne sont utilisées pour prédire le niveau d'effort requis pour la modification d'un produit logiciel. " [ISO00b]

3.2.1 Évolution des logiciels

Nos décisions de maintenance sont assistées par la compréhension de ce qui se déroule dans le système au travers du temps. Selon les observations de Lehman sur le comportement des systèmes, l'évolution des programmes respecte cinq lois [Leh80, p. 9] :

1) Modifications perpétuelles Un programme qui est utilisé subit le changement incessant ou devient progressivement moins utile. Le processus de changement ou de dégradation continue jusqu'à ce qu'il soit plus coûteux que le remplacement du système par une version recréée.

2) Complexité croissante Étant donné qu'un programme qui évolue est continuellement modifié, sa structure se détériore. En reflétant cela, sa complexité augmente à moins qu'un travail ne soit fait pour la maintenir ou la réduire.

3) Loi fondamentale de l'évolution des programmes L'évolution des programmes est sujette à une dynamique qui crée les processus de programmation, et donc des mesures du projet global et des attributs du système, s'auto-régulant. Cette auto-régulation suit des tendances statistiques déterminables et des invariances.

4) Conservation de la stabilité organisationnelle Lors de la vie active d'un programme, le taux d'activité de maintenance dans un projet de programmation est statistiquement invariant.

5) Conservation de la familiarité Lors de la vie active d'un programme, le contenu des modifications (modifications, ajouts, suppressions) d'un programme évoluant est statistiquement invariant.

La loi des modifications perpétuelles implique que la maintenance est inévitable et qu'un grand système n'est jamais complet. La loi de complexité croissante dit qu'à moins d'une intervention pour réduire la complexité, celle-ci ne cessera pas de croître. Selon la loi fondamentale, les systèmes logiciels manifestent des comportements et des tendances qui sont mesurables et prédictibles. La conservation de la stabilité organisationnelle exprime qu'il n'existe pas de fluctuation majeure des attributs organisationnels. De façon similaire, la dernière loi déclare qu'après un moment, les effets des délivrables suivants provoquent une légère différence sur l'intégralité des fonctionnalités du système [PA09, p. 567].

3.2.2 Processus de maintenance

Belady et Lehman [BL72] sont les premiers chercheurs à essayer de capturer l'effort de maintenance dans un modèle prédictif. Ils tiennent compte de la détérioration qui survient au travers du temps dans un grand système. Dans les très grands systèmes, les mainteneurs doivent devenir experts dans certains aspects du système. C'est-à-dire que chaque membre doit se spécialiser dans une fonction particulière ou dans un domaine de performance : base de données, interface utilisateur ou réseau logiciel, par exemple. Cette spécialisation laisse parfois l'équipe sans aucun généraliste ; c'est-à-dire qu'il n'existe pas de personne avec une perspective du système dans son ensemble. La spécialisation de l'équipe mène généralement à une augmentation exponentielle des ressources

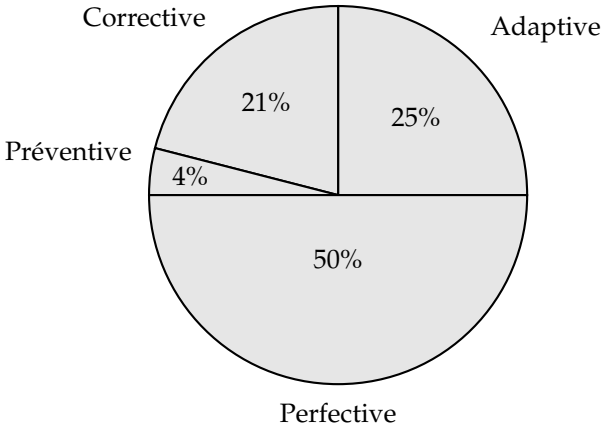


FIGURE 3.4 – Distribution de l'effort de maintenance [LS81]

consacrées à la maintenance [PA09, p. 578].

La maintenance se concentre simultanément sur quatre aspects majeurs de l'évolution d'un système :

1. Maintenir le contrôle sur les fonctions du système au jour le jour
2. Maintenir le contrôle sur les modifications du système.
3. Perfectionner les fonctions existantes
4. Prévenir les performances du système de la dégradation jusqu'à des niveaux inacceptables

Chacun de ces points provoque une activité de maintenance [PA09]. Ces quatre activités sont réparties statistiquement en terme d'effort selon la Figure 3.4 de Lientz et Swanson [LS81]. Leur enquête porte sur 487 projets logiciels en maintenance.

- La **maintenance corrective** s'adresse au problème de fautes relatives au contrôle des fonctions au jour le jour.
- La **maintenance adaptative** est l'implémentation de modifications nécessaires au bon fonctionnement d'une autre modification récemment introduite dans le système.
- La **maintenance perfective** implique des changements pour améliorer certains aspects du système, même si les modifications ne sont pas sujettes aux fautes (améliorer la conception d'une fonction récemment introduite, par exemple).
- La **maintenance préventive** est similaire à la maintenance perfective, elle implique la modification de certain aspects dans le but de prévenir des fautes (par exemple, l'ajout de vérifications de types ou l'amélioration de la gestion d'erreurs).

Par exemple, Nguyen et al. proposent une expérience où ils confrontent un ensemble d'étudiants à un programme écrit en C++ sur lequel ils doivent réaliser des tâches de maintenance. En terme d'activités de maintenance, ils se concentrent sur des tâches rapides en fonction des requêtes de maintenance d'un client. Les auteurs de cette expérience groupent les scénarios de tâches en quatre activités de maintenance [NBD09] :

1. La **compréhension de la tâche** inclut la lecture, la compréhension des spécifications de la tâche et la demande d'informations supplémentaires.
2. L'**isolation** implique la localisation et la compréhension des segments de code à adapter.
3. L'**édition du code** inclut la programmation et le débogage du code affecté.
4. Le **test unitaire** implique l'accomplissement des tests sur le code affecté.

Le critère de qualité d'analysabilité interne, selon le standard ISO/IEC 9126, mesure donc la facilité de compréhension des programmes. Cet aspect est notamment traité par la théorie nommée "Théorie de la compréhension de programme" présentée dans la section suivante.

3.2.3 Définition

Le standard ISO/IEC9126 propose cinq types de métrique qui composent la maintenabilité interne [ISO00b] :

1. Les métriques d'**analysabilité** interne indiquent un ensemble d'attributs pour prédire l'effort dépensé par le mainteneur ou l'utilisateur ou leurs ressources consommées en essayant de diagnostiquer les déficiences ou les causes de fautes, ou pour l'identification de parties à modifier dans un produit logiciel.
2. Les métriques de **changeabilité** interne indiquent un ensemble d'attributs pour prédire l'effort dépensé par le mainteneur en essayant d'implémenter une modification spécifique dans le produit logiciel.
3. Les métriques de **stabilité** interne indiquent un ensemble d'attributs pour prédire à quel point un certain produit logiciel serait stable après une modification.
4. Les métriques de **testabilité** interne indiquent un ensemble d'attributs pour prédire la quantité de conception et d'implémentation de tests autonomes à mettre en oeuvre pour les fonctions présentes dans le produit logiciel.
5. Les métriques de **conformité** interne liées à la maintenabilité indiquent un ensemble d'attributs pour l'évaluation de la capacité du produit logiciel à se conformer à des éléments tels que des standards et des conventions. La conformité est une caractéristique relative à tous les facteurs de qualité de ISO/IEC 9126 [ISO00b].

Selon Pfleeger et al. [PA09, p. 501], la maintenabilité logicielle est définie comme étant la probabilité que, pour une condition d'utilisation donnée, une activité de maintenance puisse être menée à bien dans un intervalle de temps fixé en utilisant des procédures et des ressources fixées.

Ci-dessus, nous mentionnons la maintenabilité *interne*, qui se trouve être la caractéristique de maintenabilité à laquelle nous nous intéressons.

CHAPITRE 4

Sciences cognitives et compréhension des programmes

> L'ordre règne dans les gènes pendant que les défenseurs des sciences cognitives s'efforcent de découvrir dans le cerveau l'ordinateur de leurs rêves.
>
> Jean-Didier Vincent

FACE à des tâches de maintenance, chaque développeur réagit différemment. Néanmoins, plusieurs auteurs tentent de généraliser le processus cognitif de ces mainteneurs lors de la phase dite de compréhension. Cette phase est indispensable et tout développeur réalisant une activité de maintenance doit y prendre part.

Alors que la Section 4.1 décrit la théorie qui généralise le processus mental d'acquisition d'information d'un ingénieur du logiciel face à un programme, la Section 4.2 décrit une autre théorie qui expose les connaissances relatives au processus de vision des humains. Ces deux théories sont ensuite unifiées dans la Section 4.3 afin de permettre l'étude de la compréhension d'un programme par un développeur à l'aide d'informations relatives à sa vision.

Les Sections 4.1, 4.2 et 4.3 reprennent des matériaux publiés dans notre article se trouvant en Annexe B. Elles s'inspirent globalement du livre de Palmer [Pal99] et des travaux de Guéhéneuc [Gué09].

Sommaire

4.1 Théories en compréhension de programmes

La problématique de la compréhension de programmes est au coeur de l'analyse de la maintenabilité des logiciels. Afin de pouvoir effectuer des opérations de maintenance sur un programme, il est essentiel d'en saisir tous les aspects nécessaires à sa modification. Plus un programme est compréhensible, plus il est maintenable, c'est-à-dire facile d'y implémenter de nouvelles fonctionnalités ou de l'adapter aux besoins de l'utilisateur. De nombreux scientifiques se sont intéressés à la problématique de la compréhension de programmes. Cette section présente les grandes lignes des principales théories existantes concernant la compréhension des programmes.

La littérature sur la compréhension de programmes se concentre sur les problèmes d'obtention de données à partir d'artefacts logiciels tels que des données statiques et dynamiques, des fonctionnalités, de la documentation et d'autres informations (par exemple [AHU74, Spi03]). Elle offre des moyens pour représenter et pour communiquer ces données en utilisant des techniques variées à partir des éditeurs de texte jusqu'aux environnements 3D dynamiques, tels que [SSL01]. Cette littérature est essentielle afin d'appréhender les types de données à la disposition des ingénieurs du logiciel pour comprendre les programmes.

4.1.1 Théories générales

Storey [Sto05] propose une synthèse des théories de compréhension de programmes qui ont émergé durant les 30 dernières années. Une des premières théories, proposée par Brooks [Bro78], décrit la compréhension de programmes comme un processus de construction d'une séquence de **domaines de connaissance** afin de combler les fossés entre les domaines de problèmes et l'exécution des programmes. Une succession de domaines de connaissance décrit la compréhension d'un programme par un ingénieur logiciel.

Une autre théorie, développée par von Mayrhauser [vMV95], est une théorie intégrée décrivant le processus mis en place dans l'esprit des ingénieurs logiciels. Elle représente ce processus comme une combinaison de processus de compréhension *bottom-up* et *top-down*, travaillant avec une base commune de connaissances. Cette théorie intégrée tient compte de la dynamique de la formation et de l'abstraction des représentations mentales d'un programme.

Des travaux étudient également les "contextes" dans lesquels l'activité de compréhension de programmes prend place. Murphy et al. [MKRv05] distinguent, décrivent et identifient les tendances récurrentes dans les activités quotidiennes de l'ingénierie du logiciel. Bien que non explicitement lié à la théorie présentée dans cette section, leur travail apporte des éclaircissements à l'activité de compréhension de programmes car cette phase fait partie intégrante de toute activité d'ingénierie du logiciel. De ce fait, cette ligne de recherche est importante pour généraliser et confirmer les hypothèses émises en théorie de compréhension de programmes.

4.1.2 Recherches liées à l'usage du diagramme

Tout développeur utilise des diagrammes comme moyen graphique de transmettre des informations à d'autres développeurs ou pour mieux comprendre les programmes. Les diagrammes réduisent l'effort d'apprentissage et de compréhension en omettant les détails non pertinents et en mettant en exergue les informations pertinentes. L'hypothèse sous-jacente est que plus l'information présentée sur un diagramme est proche de la représentation mentale du développeur, plus il est

facilement compréhensible [CK05].

Les diagrammes de classes ont été largement étudiés dans la littérature sur la compréhension des programmes. Ils représentent la structure et le comportement global [JW99] des programmes, montrant les classes, interfaces et leurs relations. Ils sont souvent utilisés par les ingénieurs logiciels durant le développement et la maintenance pour abstraire les détails d'implémentation et présenter une version plus claire et plus facilement compréhensible du code source [Gué04, Raj02, SW05]. Nous rappelons seulement ici les grandes lignes de l'état de la recherche dans le domaine de la compréhension de programmes utilisant des diagrammes de classes.

Purchase et al. [PAC02] présentent les résultats de leurs expériences sur les effets du critère esthétique des préférences des utilisateurs de diagrammes de classes UML. Ils effectuent plusieurs expériences avec des sujets afin d'évaluer les préférences de ceux-ci du point de vue esthétique. Ils ont collecté des données quantitatives sous forme de pourcentage de sujets préférant un diagramme à un autre. Ils ont également collecté des données qualitatives pour chaque diagramme. Leurs conclusions montrent que le critère esthétique le plus important pour les diagrammes de classes UML sont les arcs joignant les héritages et les indicateurs de direction des arcs.

Eichelberger, dans [Eic03], étudient la relation entre la notation UML des diagrammes de classes, les principes des interactions homme-machine et les principes de programmation et de conception orienté objet. L'auteur suggère de changer la notation UML et les critères esthétiques afin d'étendre les diagrammes de classes. Ces changements et les critères esthétiques sont implémentés dans un outil, *Sugi*, pour étendre des diagrammes semblables aux diagrammes de classes. L'auteur prétend que réorganiser les diagrammes de classes conformément aux critères esthétiques augmente la lisibilité des diagrammes. Néanmoins, seuls des arguments qualitatifs ont été fournis.

Hadar et Hazzan [HH04] présentent les résultats d'une étude sur les stratégies appliquées par les ingénieurs logiciels dans le processus de compréhension des modèles visuels des programmes. Ils utilisent des modèles visuels en utilisant la notation UML incluant les cas d'utilisation, les diagrammes d'activité, de classes, de séquences, de collaborations, d'états, d'objets, de paquetages et de déploiement. Les sujets étaient des étudiants en informatique de plusieurs universités. Ceux-ci ont été divisés en deux groupes pour collecter des données qualitatives et quantitatives. Les auteurs montrent l'utilité des différentes facettes de description de programmes fournies par UML et qu'aucun type de diagramme n'est plus important qu'un autre. Cependant, des études futures sont nécessaires pour confirmer ces découvertes.

Sun et Wong évaluent dans [SW05] les algorithmes d'arrangement pour les diagrammes de classes de deux outils industriels : Rational Rose et Borland TogetherPurchase. Cette évaluation est faite selon le critère de travaux précédents, incluant les travaux cités par Purchase et al. [PAC02] et Eichelberger [Eic03]. En utilisant les lois de la théorie de la Gestalt de la perception visuelle [Pal99, p. 50–53], ils retiennent et justifient 14 critères pour évaluer la qualité de la disposition des diagrammes de classes. Ils ont appliqué ces critères sur un programme de thermomètre et sur JUnit [GB98]. Ils concluent sur la bonne qualité des deux outils industriels, sur la pertinence de leur critère et sur la difficulté de satisfaire tous les critères.

4.1.3 Recherches liées à l'aspect visuel

Les aspects visuels tels que la disposition ou les formes d'une ressource sont des éléments à prendre en compte lors de l'analyse du processus d'acquisition de l'information. De ce fait, certaines expériences ont été réalisées afin de montrer l'influence de l'aspect visuel des diagrammes sur la compréhension de ceux-ci.

Guéhéneuc [Gué06] mène une expérience avec un oculomètre pour étudier comment les ingénieurs logiciels assimilent et utilisent l'information de diagrammes de classes UML. Il montre l'importance des classes, des interfaces et que les développeurs semblent peu utiliser les relations binaires entre classes comme l'héritage ou la composition. Yusuf et al. [YKM07] mènent une expérience similaire pour analyser l'utilisation de caractéristiques spécifiques des diagrammes de classes UML (par exemple, la disposition, la couleur et les stéréotypes) durant la période de compréhension du programme. Ils mettent en évidence l'efficacité de la disposition des éléments avec des informations supplémentaires tels que les couleurs ou les stéréotypes pour améliorer la compréhension du programme.

Bednarik et Tukiainen [BT06] proposent une approche pour étudier les tendances basées sur des mesures effectuées par un oculomètre. Ces données étaient récoltées en observant des sujets effectuant des petites tâches de maintenance. En utilisant cette approche, ils caractérisent les stratégies de compréhension de programmes en utilisant différentes représentations de programmes (lecture du code et exécution du programme).

Sharif et al. [SM10] étudient si l'utilisation de la convention "camelCase" (utilisation de majuscules dans les noms d'identificateurs pour séparer les mots sans y mettre d'espaces) permet une meilleure compréhension du code que l'utilisation d'*underscores*. Ils répliquent une étude précédente de Binkley et al. [BDLM09] qui montrait que l'utilisation du "camelCase" permet une meilleure précision dans la lecture du code. Étonnamment, ils montrent des résultats opposés à cette étude : bien que les données n'ont indiqué aucune différence dans la précision entre les identifiants utilisant "camelCase" et les *underscores*, les sujets ont reconnu plus efficacement les identificateurs utilisant les *underscores*.

Uwano et al. [UNMiM06] étudient les habitudes de lecture du code des développeurs et identifient les patrons qui distinguent les lecteurs "efficients" des autres. Ils ont préalablement implémenté un environnent intégré pour mesurer et enregistrer les mouvements des yeux des lecteurs et, basé sur leurs fixations et ont identifié les lignes du code source que les développeurs lisent. Ils ont mené une étude empirique sur 30 processus de relecture de six programmes par cinq relecteurs. Ils ont montré que tous les relecteurs parcouraient le code source en suivant un patron similaire et que ceux qui n'avaient pas passé assez de temps à "scanner" le code avaient tendance à prendre plus de temps à trouver les erreurs dans le code.

Navarro-Prieto et Canas [NP98] expérimentent pourquoi et sous quelles circonstances les langages de programmation visuels seraient plus facilement compréhensibles que les langages de programmation textuels. Ils expliquent que les recherches en psychologie et en traitement de l'information visuelle montrent que le traitement d'une image permet un accès plus rapide à l'information sémantique. En conséquence, les langages de programmation visuels devraient permettre une construction du modèle mental plus rapide que les langages procéduraux. Selon leur conclusion, les utilisateurs

de tableurs développent des représentations mentales basées sur les flux de données alors que les programmeurs C semblent accéder en premier lieu à une représentation mentale basée sur le contrôle de flux [NP98].

4.2 Théories de la vision

La science de la vision est le domaine des sciences qui s'intéresse aux systèmes de vision des humains. Cette science collecte des faits sur la vision, formule des lois à partir de ces faits et invente des théories qui expliquent ces lois et ces faits. Avec ces théories, les scientifiques de la vision ont été capables de prédire de nouveaux faits avec succès et de raffiner leurs théories.

La science de la vision possède nombre de théories pour expliquer la vision des couleurs, la vision de l'espace, la perception des mouvements et des évènements, tout autant que les mouvements des yeux, la mémoire visuelle et la conscience visuelle. Du mieux que nous sachions, le livre de Palmer présente la couverture la plus complète et la plus approfondie des théories de la vision, converti dans le paradigme du traitement de l'information (*information processing paradigm*) [Pal99].

4.2.1 Lien entre vision et processus cognitif

Afin de pouvoir utiliser un système oculométrique et de tirer des conclusions sur la charge de travail en se basant sur les mesures effectuées par cet appareil, il est nécessaire de bien saisir le lien entre vision et processus cognitif. Le fait que notre vision ne nous permette de voir très nettement que ce qui se trouve au centre de notre vision (sur la fovéa [1]) permet de s'assurer que le sujet centrera son regard sur l'objet de l'attention afin de l'analyser. Il est en effet très difficile et contre-intuitif d'observer un objet sans déplacer celui-ci au centre de la fovéa par un mouvement des yeux ou de la tête. En suivant le mouvement des yeux d'un individu, il est donc possible de connaître l'objet de son attention [Duc07].

Le chemin du parcours visuel sur une image n'est pas dépendant de l'image, mais bien de la tâche qui est à accomplir. Effectivement, un sujet ne regardera pas les mêmes éléments d'une photo si nous lui demandons de compter le nombre de personnes présentes dans une pièce ou bien de trouver la source de lumière de l'image, par exemple. Par conséquent, nous remarquons ici qu'il est possible d'estimer si une personne observe les bons éléments d'un diagramme, par exemple, en fonction de la tâche qui est demandée. Notons également que l'attention est fonction de l'individu. Par exemple, il est plus facile pour une personne de repérer son nom dans un texte. Nous pouvons y voir ici un parallèle avec l'attention auditive lorsque nous entendons notre nom (effet *cocktail party*). Le principe d'attention n'est donc pas inné.

La Figure 4.1 illustre clairement cet effet. Dans cette expérience, il est demandé à un sujet de parcourir l'image afin de répondre à plusieurs questions. Les questions étaient : (1) examiner l'image sans aucune contrainte, (2) estimer le revenu économique des personnes présentes sur l'image, (3) estimer l'âge des personnes, (4) estimer ce que faisaient les personnes avant l'arrivée du visiteur, (5) se souvenir des vêtements portés par les personnes, (6) se souvenir de la position des personnes et des objets dans la pièce, (7) estimer le temps depuis la dernière venue du visiteur. Les parcours visuels correspondant à ces sept tâches sont différents.

1. La fovéa est un zone riche en photo-récepteurs située au centre de la rétine.

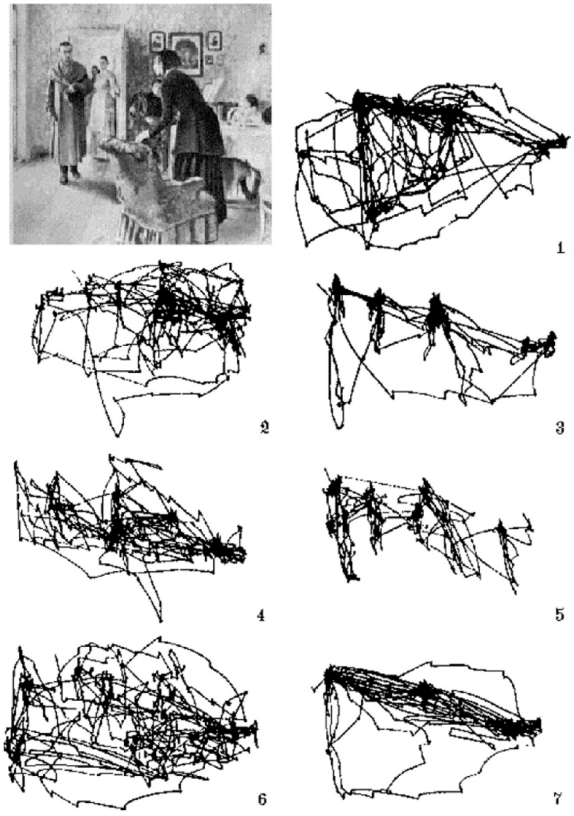


FIGURE 4.1 – Comparaison du parcours visuel d'un individu en fonction de la tâche à accomplir (tiré de [Pal99])

4.2.2 Types de mouvements oculaires

Le champ visuel d'une personne moyenne est de 130 degrés verticalement et 180 degrés horizontalement. Bon nombre de nos mouvements oculaires sont totalement inconscients (par exemple, les Nystagmus physiologiques, mouvements involontaires et saccadés du globe oculaire [Pal99, p. 521]) et servent simplement à positionner le regard sur le stimuli, ou sur l'objet sujet de notre attention.

Il existe deux fonctions principales du mouvement des yeux : amener ou garder l'objet d'intérêt sur la fovéa et stabiliser l'image sur la rétine lorsque le sujet est en mouvement. La première fonction comprend les saccades, vergences, poursuites et fixations. La seconde comprend les réflexes vestibulo-oculaires et les réflexes optocinétiques [RT06].

Saccades Les saccades sont des mouvements rapides de l'œil qui se produisent souvent lorsque nous regardons les alentours d'un environnement stationnaire (par exemple, le mouvement de lecture d'un texte). Ce mouvement est un mouvement "balistique". Une fois commencé, il ne peut être arrêté. Si l'œil rate sa cible, une autre saccade peut toujours être faite pour résoudre le problème, mais une fois qu'une saccade est entamée, sa destination semble fixée. Une saccade prend 120-150ms à être planifiée et exécutée. Le fait que nous ne percevions pas de "flou" lors de ces mouvements est dû au fait que la perception semble être atténuée durant le mouvement, il s'agit d'une "*saccadic supression*" [Pal99]. Peu, voire aucune analyse visuelle ne peut être effectuée durant une saccade étant donné la trop grande rapidité du mouvement. Les saccades sont souvent ignorées si ce n'est pour retracer le parcours visuel d'un individu [SG00].

Vergence "Mise au point" de la vision lors du mouvement d'un objet observé en profondeur.

Poursuite oculaire Une poursuite oculaire permet de maintenir l'objet de l'attention sur la fovéa si l'objet se déplace. Elle est utilisée pour suivre la position d'un objet en déplacement afin de le garder dans la vision focale.

Fixation En science de la vision, une fixation est la position de l'œil sur l'objet de l'attention et permet à celui-ci de rester sur la fovéa lorsqu'il est stable [Pal99]. Le centre d'une fixation typique est dans les 2-3° de la cible observée [Rob79] et le temps minimum nécessaire d'une fixation pour procéder à l'analyse visuelle est de 100-150ms [Viv90]. Les fixations sont souvent perturbées par des micro-mouvements tels que les glissades, tremblements et micro-saccades mais ce sont ces micro-mouvements qui permettent de caractériser les fixations. S'il n'existe pas de perturbations telles que des mouvements de la cible ou de l'individu, les fixations se feront sur l'objet de l'attention. Le fait que les fixations soient caractérisées par des mouvements peut sembler contre-intuitif mais ceux-ci sont en fait nécessaires à la vision. En effet, sans ces micro-mouvements, l'image disparaîtrait sur la rétine après une à deux minutes. La durée d'une fixation est de 150 à 600ms et les fixations constituent 90% du temps de visualisation [Duc07, p 46]. Le processus cognitif est effectué durant les fixations. Salvucci et Goldberg [SG00] et Shic et Chawarska [SSC08] étudient en profondeur les différents algorithmes permettant de différencier les fixations des saccades.

Réflexe vestibulo-oculaire et optocinétique Ensemble, ces réflexes permettent de maintenir l'objet de l'attention au centre de la fovéa et ce, nonobstant les mouvements de rotation de la tête ou le déplacement du corps.

Il existe de nombreux moyens de mesurer tous ces mouvements, ces moyens font l'objet de la section suivante et de la Section 8.1.

4.2.3 Oculométrie

L'oculométrie (*eye-tracking*) est une technique qui mesure l'endroit où un individu pose son regard [PB06]. Ces mesures permettent à un expérimentateur de connaître l'endroit où une personne regarde à n'importe quel moment mais également de connaître le parcours visuel qu'a effectué un individu sur une image, par exemple.

Histoire de l'oculométrie

L'oculométrie est utilisée dans de nombreux domaines (publicité, sciences, aviation, conduite automobile…) mais a récemment été très utilisée dans le domaine de l'évaluation d'interfaces graphiques (qu'il s'agisse des sites Web ou des interfaces de programmes). Les observations du mouvement des yeux peuvent, par exemple, permettre aux scientifiques d'estimer l'intuitivité d'une interface. Les recherches actuelles tendent à utiliser les systèmes d'oculométrie comme entrée utilisateur pour un système donné. Il est déjà possible, par exemple, de contrôler le curseur d'un ordinateur uniquement grâce aux mouvements des yeux.

Plusieurs méthodes ont été utilisées dans le passé pour tenter de mesurer les mouvements des yeux. Les premiers pionniers dans ce domaine datent de plus d'un siècle [PB06]. Les mesures électro-oculographiques, par exemple, reliaient des électrodes posées sur la peau autour des yeux et pouvaient mesurer les différences de potentiel électrique et donc de détecter les mouvements des yeux. D'autres appareils demandaient l'utilisation de larges lentilles de contact qui couvraient la cornée [Pal99]. Ces méthodes étaient relativement invasives et peu précises. Les système modernes utilisent des images vidéos de l'œil pour déterminer où une personne porte son regard.

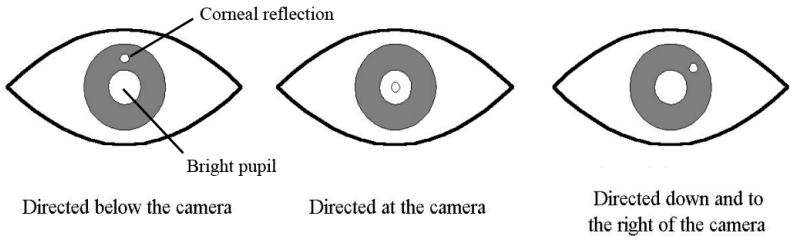


FIGURE 4.2 – Position de la réflexion de la cornée en fonction du point de regard (cf. [RRCL01])

Fonctionnement d'un oculomètre

La majorité des systèmes de mesures oculométriques disponibles sur le marché utilisent la méthode de mesure "réflexion-cornée/centre de la pupille". Ces systèmes utilisent une source de lumière infra-rouge (pour éviter les reflets dûs à la lumière naturelle) et une ou plusieurs caméras infra-rouge. La lumière infra-rouge crée un net reflet sur l'œil du sujet et rend le suivi de cette réflexion plus facile. Une fois que le logiciel a déterminé le centre de la pupille et la réflexion de la cornée, il est possible (via des calculs trigonométriques) de déterminer le point de regard. Il est également possible d'estimer l'endroit du regard "manuellement" comme le montre la Figure 4.2 [PB06].

Les systèmes de mesures oculométriques ont besoin d'être finement configurés et calibrés en fonction de la personne dont les yeux sont observés. Cette étape de calibration est critique et est effectuée à l'aide d'une série de points sur l'écran que l'utilisateur doit fixer les uns après les autres. Cette étape permet au logiciel de mesurer les distances pour chaque position du regard et les associer à un couple de coordonnées (x, y). La Figure 11.2 est un exemple d'une telle grille de calibration.

Les micro-mouvements des fixations décrits en Section 4.2.2 sont souvent considérés comme du bruit par la plupart des oculomètres. Les fixations peuvent donc sembler peu précises, mais il s'agit en fait d'un rayon de fixations étant donné que les micro-mouvements aléatoires se situent dans un angle inférieur à cinq degrés [Duc07].

Mesures oculométriques

Cette section présente différentes mesures et variables qu'il est possible de calculer à partir des données collectées par un oculomètre classique. De nombreuses métriques oculométriques existent, le but de cette section n'est pas d'être exhaustive, mais de présenter les mesures les plus souvent étudiées en génie logiciel expérimental. Les métriques oculométriques les plus souvent utilisées sont résumées dans [JK58, p. 582]. Il est toutefois nécessaire de préalablement définir les différentes notations utilisées dans cette section :

A **:** ensemble de réponses.

F **:** ensemble de fixations.

AOI_i **:** ensemble de zones d'intérêt pour la question i ($AOI_i = AORI_i \cup AOII_i$).

$AORI_i$ **:** ensemble des zones d'intérêt pertinentes pour la question i.

$AOII_i$ **:** ensemble des zones d'intérêt non-pertinentes pour la question i.

$t : F \rightarrow \mathbb{N}$ **:** La durée d'une fixation spécifiée (en millisecondes).

$fix : AOI \times A \rightarrow F^*$ **:** Les fixations dans une zone d'intérêt spécifique pour une réponse particulière.

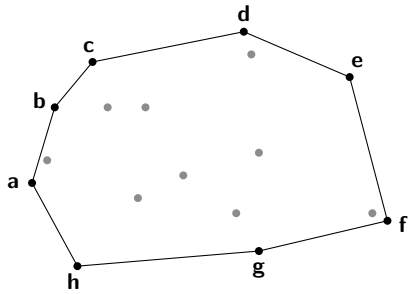


FIGURE 4.3 – Enveloppe convexe inspirée de [HPS08, p. 257]

$surface : AOI \rightarrow \mathbb{N}$ **:** La surface de la zone d'intérêt spécifiée (en pixels^2).

$FAORI_i$ **:** L'ensemble des fixations dans une zone d'intérêt pertinente pour la réponse i ($i \in A_j$) tel que :

$$FAORI_i = \bigcup_{k \in AORI_j} fix(k, i).$$

$FAOII_i$ **:** L'ensemble des fixations dans une zone d'intérêt non-pertinente pour la réponse i ($i \in A$) tel que :

$$FAOII_i = \bigcup_{k \in AOII_j} fix(k, i).$$

$TAOI_i$ **:** Durée totale (en millisecondes) des fixations dans une zone d'intérêt pour la réponse i ($i \in A_j$) :

$$TAOI_i = TAORI_i + TAOII_i.$$

$TAORI_i$ **:** Durée totale (en millisecondes) des fixations dans une zone d'intérêt pertinente pour la réponse i ($i \in A_j$) :

$$TAORI_i = \sum_{f \in FAORI_i} t(f).$$

$TAOII_i$ **:** Durée totale (en millisecondes) des fixations dans une zone d'intérêt non-pertinente pour la réponse i ($i \in A_j$) :

$$TAOII_i = \sum_{f \in FAOII_i} t(f).$$

$\#F_i$ **:** Le nombre total de fixations pour une réponse i d'un sujet ($i \in A$).

Zone d'intérêt Une zone d'intérêt est une zone visuelle caractérisée par une certaine pertinence. Elle est considérée par l'expérimentateur comme pertinente pour une tâche si le sujet doit regarder cette zone pour répondre à la tâche. Une zone peut également être non-pertinente ou bien peut être ignorée (le cas, par exemple, des zones se situant en dehors de la donnée visuelle).

Plusieurs métriques sont décrite ci-dessous.

Enveloppe convexe Pour un ensemble de points P dans un plan à deux dimensions, l'enveloppe convexe est l'ensemble convexe le plus petit qui contient tous les points de P (cf. Figure 4.3) [HPS08, p. 257]. L'enveloppe convexe permet de mettre la zone couverte par le parcours visuel d'un sujet en évidence en considérant les fixations comme des points. Plus la surface de l'enveloppe convexe est faible, plus la recherche du parcours visuel est efficiente.

Nombre moyen de fixations Cette métrique est basée sur le nombre de fixations par zone d'intérêt et a été utilisée par Goldberg et al. :

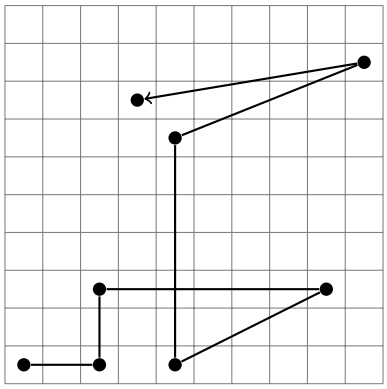


FIGURE 4.4 – Exemple de densité spatiale égale à 8%

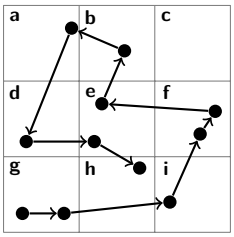


FIGURE 4.5 – Exemple de parcours visuel

	a	b	c	d	e	f	g	h	i
a				1					
b	1								
c									
d					1				
e		1						1	
f					1	1			
g							1		1
h									
i						1			

TABLE 4.1 – Matrice de transition de la Figure 4.5

> "*L*e nombre total de fixations est corrélé négativement avec l'efficience de la recherche. " [GK99]

Le nombre de fixations par zone d'intérêt indique que certaines zones sont plus remarquables ou plus importantes que d'autres [PB04]. Selon Just et al. [JC76], la durée d'une fixation sur une zone peut avoir deux significations différentes :

1. Le sujet a eu du mal à extraire l'information [FJM50].
2. Le sujet est plus attiré par l'objet ("*more engag[ed] in some way" by the object*) [PB06].

Densité spatiale La densité spatiale est une métrique très souvent utilisée. La densité spatiale d'un parcours visuel est présentée comme exemple sur la Figure 4.4. Dans cet exemple, 8 cellules de la grille visuelle sont parcourues, la densité spatiale est donc de 8%.

> "*L*a couverture (les différentes parties parcourues lors de la recherche par un individu) d'une image peut être mesurée par la distribution spatiale des points du regard. [...] L'interface [(ou l'image)] peut être divisée en zones selon une grille, chacune associée à un objet spécifique ou physique sur l'écran. [...] La *densité spatiale* est égale au nombre de cellules contenant au moins une fixation, divisé par le nombre total de cellules.[...] Plus la densité spatiale est faible, plus la recherche est dirigée, peu importe l'ordre dans lequel les fixations ont été enregistrées. " [GK99]

Matrice de transition Il s'agit d'une matrice (proposée par [PSF95]) représentant le passage du regard d'une région à une autre.

> "*L*es transitions fréquentes d'une région de l'affichage à une autre indiquent l'inefficacité d'un balayage avec une vaste recherche. [...] La matrice de transition est

une représentation en tableau du nombre de transitions, pour chaque zone, d'une zone à une autre. " [GK99]

La Figure 4.5 illustre un exemple de parcours visuel sur une grille d'affichage et le Tableau 4.1 représente sa matrice de transition. Une cellule en position (x, y) qui contient la valeur `1` signifie qu'une transition existe d'une cellule x à une cellule y. Les seules valeurs disponibles pour ces cellules sont `1` ou `0`.

Densité de transition La densité de transition est basée sur la notion de matrice de transition et est calculée de la façon suivante :

$$TRANSITION\_DENSITY = \frac{\sum_{x \in C} isFilled(x)}{\#C}$$

où C est l'ensemble des cellules dans la matrice de transition et $isFilled : C \rightarrow \{0, 1\}$ retourne 1 si la cellule spécifiée est remplie et 0 sinon. Deux chemins visuels peuvent avoir la même enveloppe convexe et même densité spatiale, mais une densité de transition différente. Un meilleur chemin visuel aurait une densité de transition plus faible.

Taux normalisé de fixations pertinentes Jeanmart [Jea08] suggère d'utiliser cette métrique définie comme le ratio entre le nombre normalisé de fixations par zone d'intérêt pertinente et le nombre normalisé de fixations dans les zones d'intérêt non pertinentes. Cette variable permet de représenter un effort. Un taux élevé correspond à un sujet réalisant peu d'effort et un taux faible correspond à un sujet réalisant beaucoup d'effort. Cette métrique est une version simplifiée de la métrique *IN AORI / IN AOII normalisé* détaillée ci-dessous.

IN AORI / IN AOII Ce ratio est calculé sur base du nombre de fixations dans les zones d'intérêts pertinentes divisé par nombre de fixations dans les zones d'intérêts non-pertinentes. Cette métrique ignore toutes les fixations qui sont hors zone d'intérêt. Ce rapport est un indicateur de la pertinence de l'effort. Elle est utilisée pour évaluer la pertinence de l'effort du sujet. Le ratio IN_OUT_i pour la réponse d'un sujet i est défini tel que :

$$IN\_OUT_i = \frac{\#FAORI_i}{\#FAOII_i}.$$

IN AORI / IN AOII normalisé (par zone d'intérêt et par surface) Nous proposons cette métrique comme mesure de la pertinence de l'effort visuel d'un sujet. Elle est basée sur le principe du *Taux normalisé de fixations pertinentes* présenté par Jeanmart [Jea08] et sur la métrique *IN AORI / IN AOII*. Elle est le ratio entre le nombre de fixations dans les zone d'intérêts pertinentes (AORI) et le nombre de fixations dans les zones d'intérêts non-pertinentes (AOII) mais ces nombres sont normalisés en fonction du nombre de zones d'intérêt **mais également en fonction de leur surface**. Cette normalisation est justifiée par le fait que dans certains cas, il est tout à fait possible qu'il existe un plus grand nombre de zones d'un certain type que d'un autre. En outre, la surface d'une zone sur le diagramme peut influencer de manière significative le nombre de fixations à l'intérieur de celle-ci. Ce rapport est un indicateur de la pertinence de l'effort. Le ratio $IN\_OUT\_\text{Norm}_i$ pour la réponse d'un sujet i à une question k est défini tel que :

$$NORM\_RATE_i = \frac{\frac{FAORIs_i}{\#AORI_k}}{\frac{FAOIIs_i}{\#AOII_k}}.$$

où # retourne la cardinalité d'un ensemble, $FAORIs_i$ est le nombre de fixations dans les zones d'intérêt pertinentes normalisé par surface pour la réponse i :

$$FAORIs_i = \sum_{j \in AORI_k} \frac{\#fixj, i}{surface(j)}.$$

et où $FAOIIs_i$ est le nombre de fixations dans zones d'intérêt non-pertinentes normalisé par surface pour la réponse i :

$$FAOIIs_i = \sum_{j \in AOII_k} \frac{\#fixj, i}{surface(j)}.$$

On-target/All-target Cette métrique

> "*[...]* peut être définie en comptant le nombre de fixations tombant dans une zone d'intérêt désignée, ensuite en divisant ce nombre par le nombre total de fixations. Il s'agit d'une mesure d'efficience de recherche dépendante du contenu. Un faible ratio indique une faible efficience. " [GK99]

Cette métrique ON_ALL_ij pour une zone d'intérêt j et pour une réponse de sujet i se calcule telle que :

$$ON\_ALL_i = \frac{\#fix(j, i)}{\#F_i}.$$

Fixations Post-cible "*L*e nombre de Fixations post-cible, ou de fixations dans d'autres zones apparaissant après la visualisation de la cible peuvent indiquer la pertinence de la cible pour l'utilisateur. De hautes valeurs de recherche hors-cible apparaissant après la visualisation de la cible indiquent une cible avec une faible pertinence ou visibilité pour l'utilisateur. " [GK99]

En d'autres termes, un grand nombre de fixations hors de la zone d'intérêt visée, après qu'elle ait été fixée, indique son manque de visibilité ou son manque de mise en évidence. Cette métrique est surtout utile dans le cas d'études d'interfaces graphiques.

Oculométrie en génie logiciel

L'oculométrie a déjà été utilisée dans le domaine du génie logiciel expérimental. Par exemple, Bednarik et Tukiainen [BT04] comparent deux méthodes d'oculométrie pour évaluer le comportement des individus devant effectuer des tâches de débogage sur des programmes Java. Le premier outil était un oculomètre "classique" et le second un RFV (*Restricted Focus Viewer*). Ce dernier brouille la totalité de l'écran excepté l'endroit où se trouve le curseur de la souris. 18 participants ont participé à cette étude et l'expérience a montré qu'en terme de précision de débogage et qu'en terme de temps passé sur les différentes zones d'intérêts, le RFV n'avait pas d'influence significative. Cependant, le RFV modifie le comportement dynamique de programmation. Le nombre de passages d'une zone à une autre mesurée à l'aide du RFV diffère significativement de ceux mesurés par l'oculomètre.

Dans [BT06], Bednarik et Tukiainen utilisent de nouveau un oculomètre pour étudier la compréhension des programmes. Des programmeurs débutants et plus expérimentés ont été mis à contribution

dans cette expérience et ont utilisé un programme de visualisation pour les aider dans leur compréhension tandis que la position, la durée des fixations et les changements de zones d'intérêts étaient enregistrés. Ils ont subséquemment proposé une approche pour étudier les tendances d'observation basées sur les résultats d'oculomètres. Une fois leur approche pour caractériser les stratégies de compréhension de programmes appliquée à l'aide de données de mouvement des yeux, ils concluent que l'oculométrie peut être une source utile et précieuse dans l'analyse des processus cognitifs de compréhension de programmes.

Guéhéneuc [Gué06] mène aussi une expérience avec un oculomètre pour étudier comment les ingénieurs logiciels assimilent et utilisent l'information de diagrammes de classe UML. Il montre l'importance des classes et des interfaces et met en exergue que les développeurs (pour la tâche de compréhension) semblent peu utiliser les relations binaires entre classes comme l'héritage ou la composition.

Yusuf et al. [YKM07] conduisent une expérience similaire pour analyser l'utilisation de caractéristiques spécifiques des diagrammes de classes UML (par exemple, la disposition, la couleur et les stéréotypes) durant la période de compréhension du programme. Ils mettent en évidence l'efficacité des dispositions avec des informations supplémentaires telles que les couleurs ou les stéréotypes pour améliorer la compréhension du programme. Ils montrent également que les sujets semblent naviguer dans les diagrammes de façon différente selon leur expertise pour une tâche donnée.

Pietinen et al. [PBG+08] dirigent une expérience en utilisant un oculomètre. Cette fois-ci, des paires de sujets (développeurs) travaillent sur un seul écran. Les tâches de développement étant effectuées en même temps et en collaboration entre les deux sujets, la mesure des mouvements oculaires sont difficiles et ils font face à de nombreux problèmes d'ordre technique.

Jeanmart et al. [Jea08] étudient l'impact du patron de conception Visiteur [GHJV94] sur des tâches de compréhension et de modification de programmes sur des diagrammes UML. Trois projets *open-source* sont utilisés pour cette expérience :*JHotDraw* [2], *JRefactory* [3] et *PADL* [4]. Ils montrent que la présence du patron Visiteur et sa mise en page n'ont pas d'impact significative sur la compréhension des diagrammes de classes UML quand les sujets doivent effectuer des tâches de compréhension, mais ont une influence significative si les tâches concernent les modifications [Jea08].

Cepeda Porras [Por08] étudie l'influence des différentes méthodes de visualisation de patrons de conception sur des diagrammes UML en utilisant un oculomètre. Cette expérience lui permet de mettre en avant la méthode de visualisation la plus efficace (nécessitant la charge cognitive la plus faible).

Van den Plas [Van09] étudie l'impact du patron de conception Composite et Observateur sur des tâches de compréhension et de modification de programmes sur des diagrammes UML. Trois projets *open-source* sont sélectionnés pour cette étude : JUnit [5], QuickUML [6] et ArgoUML [7]. Trois métriques sont choisies pour évaluer l'effort des sujets : la densité spatiale, la matrice de transition et la durée moyenne des fixations dans les zones d'intérêt. L'oculomètre utilisé pour cette expérience est

2. http://www.jhotdraw.org
3. http://jrefactory.sourceforge.net/
4. http://wiki.ptidej.net/doku.php?id=padl
5. http://www.junit.org/
6. http://sourceforge.net/projects/quj/
7. http://argouml.tigris.org/

l'Eye-link®II et 24 sujets participent à l'étude. Toutefois, il n'est pas possible de tirer de résultats significatifs de cette expérience.

4.3 Théorie Vision–Compréhension

Après avoir introduit la théorie de la vision d'une part (en Section 4.2) et la théorie de la compréhension des programmes d'autre part (en Section 4.1), nous allons tenter d'unifier ces deux théories en une "théorie de la Vision–Compréhension" telle que présentée par Guéhéneuc [Gué09]. Le contenu de cette section provient en majeure partie des travaux de cet auteur.

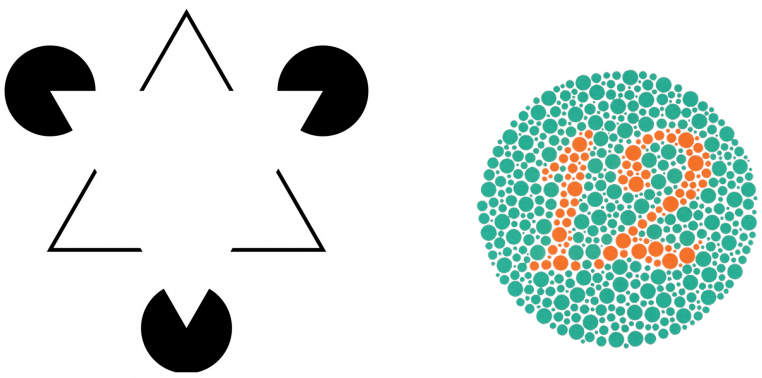
Nous avons présenté, dans les sections précédentes, différentes théories qui ont été proposées afin d'expliquer les mécanismes sous-jacents à la **compréhension de programmes**. Ces théories expliquent le processus mental que les ingénieurs logiciels mettent en place pour effectuer des tâches sur l'information qu'ils ont acquise. Ces théories tentent également d'expliquer les processus mentaux nécessaires pour comprendre un programme. Cependant, ces théories n'expliquent pas *comment* cette information est acquise. La **théorie de la vision** énonce des théories sur les processus mis en place par les personnes pour acquérir de l'information visuelle de leur environnement. Joindre la science de la vision et la compréhension de programmes dans un *framework* théorique permet d'expliquer certains phénomènes de la compréhension de programmes, de prédire de nouveaux phénomènes et d'imaginer de nouvelles expériences.

Guéhéneuc décrit dans son article [Gué09] que les ingénieurs logiciels utilisent tous leurs sens et leurs capacités cognitives lors de la phase de compréhension de programmes et en particulier la vue. En effet, ceux-ci lisent de la documentation et le code source des programmes pour créer leur modèle mental du programme. La vue est donc un élément clé dans la compréhension des programmes. À sa connaissance, aucune théorie de compréhension de programmes ne reconnaît explicitement l'importance de la vue.

Par conséquent, il propose cette théorie qui met l'accent sur l'acquisition de l'information visuelle par les ingénieurs logiciels. Cette théorie, basée sur la science de la vision, ne contredit en rien les théories émises par Brooks [Bro78] ou par von Mayrhauser [vMV95] mais les étendent en fournissant des explications quant à l'acquisition de l'information. Cette théorie peut aider à la création d'outils et d'algorithmes d'aide à la compréhensions de programme et aider à créer des expériences pour répondre aux questions liées à l'activité de compréhension de programmes au travers de l'analyse de la vue.

4.3.1 Contexte

Guéhéneuc [Gué09] développe sa théorie dans le cadre d'un besoin grandissant de compréhension des processus de compréhension de programmes pour réduire le coût de maintenance et faciliter le développement d'outils pour aider l'ingénieur logiciel à comprendre le logiciel. Cette théorie rejoint les recherches sur les alternatives en matière de visualisation graphique de modèles de programmes (comme, par exemple, les moyens de représentation en 3D [SSL01]).



(a) Motif de Kanizsa [Wik07] (b) Test d'Ishihara [Wik08]

FIGURE 4.6 – Illusions d'optique illustrant le traitement de l'information visuelle

4.3.2 Portée

La théorie définie dans cette section se limite aux ingénieurs logiciels exécutant une activité de compréhension de programmes. Les ingénieurs logiciels doivent utiliser leur vue dans des conditions normales pour appréhender le modèle du programme étudié, les autres modalités ne sont pas à prendre en compte bien qu'elles puissent influencer l'activité. Les modèles de programmes peuvent être représentés sous n'importe quelle forme, cela inclut les formes textuelles, graphiques (bi- ou tri-dimensionnelles), statiques, dynamiques ou toute autre donnée.

4.3.3 Définition

Guéhéneuc [Gué09] décompose la reconnaissance d'objets durant la phase de compréhension de programmes en différents sous-processus qui s'exécutent sur des représentations [8] différentes du modèle d'un programme. La Figure 4.7 illustre la séquence de processus et le flux de données entre eux [9].

Bien que cette théorie fournisse une explication à l'activité de compréhension, celle-ci est surtout intéressante pour lier la science de la vision et la compréhension de programmes et pour intégrer les premiers niveaux de compréhension (vision) avec les derniers niveaux (cognition). Les sous-processus, selon Guéheneuc, sont les suivants.

Image rétinienne L'activité de compréhension de programmes commence avec une image rétinienne du modèle du programme que l'ingénieur logiciel observe. L'image rétinienne peut provenir de n'importe quel modèle de programme, que ce soit une représentation textuelle, un diagramme UML, etc.

Étape basée sur l'image L'image rétinienne est analysée pour extraire les composantes spatiales de sa structure, tels que les bords, les lignes et les textures. Ce processus produit un ensemble d'éléments visuels : bords, barres et *blobs* [Mar82].

8. Le terme "représentation" est utilisé ici pour parler de la représentation cognitive du modèle d'un programme et le terme "'modèle" pour parler des modèles de programmes tels que le code source ou les diagrammes UML

9. Les boîtes constituent les représentations d'image, les cercles sont des processus, les flèches décrivent un flux d'informations, les lignes pointillées montrent les décompositions. L'absence de représentation entre les processus ne signifie pas qu'elles n'existent pas, mais simplifie le flux.

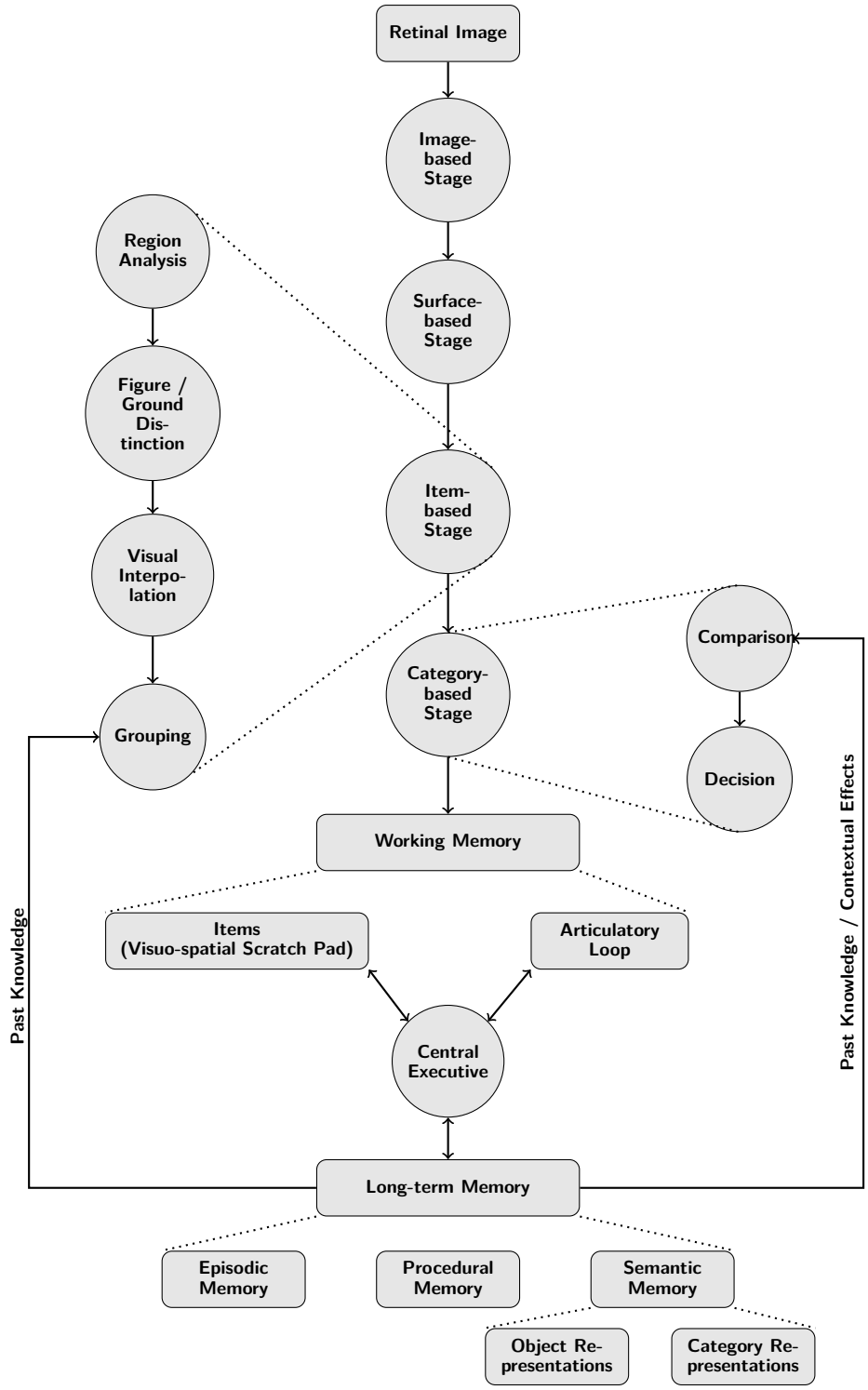


FIGURE 4.7 – Théorie de la vision-compréhension des programmes [Gué09]

Étape basée sur les surfaces Les éléments visuels du processus précédent sont interprétés en terme de surfaces visibles dans un espace 3D. Le processus produit des parties de surfaces orientées [Mar78].

Étape basée sur les objets Ce processus concerne l'organisation des éléments visuels en objets (voir [Pal99, p. 254]). Cette étape est décomposée en 4 sous-processus :

Analyse de régions Ce sous-processus analyse les éléments visuels pour identifier les régions dans la structure de l'image. Une région est une zone 2D liée qui constitue un

sous-ensemble de l'image. Différents principes régissent l'analyse de région comme la connexité, la segmentation et l'isolement des textures.

Distinction figure / fond Une fois les régions identifiées, ce sous-processus décompose les régions en deux catégories : les figures et le fond. Les figures sont des parties de l'image proches de l'observateur. Le fond est la partie de l'image loin de l'observateur, qui s'étend derrière et ne possède pas de contour.

Intercalage visuel Ce sous-processus interpole les parties cachées des régions identifiées à partir de leurs parties visibles pour permettre au système visuel de percevoir les éléments visuels partiellement cachés. Ce processus est très bien illustré par la Figure 4.6a.

Regroupement Ce sous-processus groupe les éléments visuels ensemble pour en faire des éléments cohérents. Il utilise les lois de groupement comme la proximité, les similarités de couleur, de taille, d'orientation... C'est par exemple à cette étape qu'un ingénieur logiciel expert aura tendance à voir l'ensemble d'un patron de conception plutôt que l'ensemble des classes qui le compose. Le Test d'Ishihara [10] (cf. Figure 4.6b) illustre ce processus.

Étape basée sur la catégorie Le groupement des éléments n'est pas suffisant pour permettre la compréhension. Les éléments doivent être catégorisés pour distinguer leurs différentes fonctions. Cette étape est décomposée en deux sous-processus : la Comparaison et la Décision

Comparaison Ce sous-processus utilise les connaissances passées et le contexte pour comparer les éléments identifiés avec des éléments connus. Ce processus prend en compte le contexte dans lequel les éléments doivent être identifiés.

Décision Ensuite, ce sous-processus décide des catégories auxquelles les éléments appartiennent, typiquement en utilisant une règle de type "maximum de probabilité".

Mémoire de travail Le processus précédent produit une représentation dans laquelle les différents éléments composant le modèle du programme ont été identifiés et catégorisés. Cette représentation est gérée par la mémoire de travail, qui est semblable à la mémoire à court terme mais possède une structure interne plus importante [BH74].

Direction centrale Cet élément procède aux tâches cognitives telles que la compréhension, la résolution des problèmes et la mémorisation des tâches. Il fonctionne en étroite relation avec les deux mémoires suivantes.

Bloc-notes spatio-visuel et boucle articulatoire Le bloc-notes spatio-visuel stocke l'information visuelle et spatiale tandis que la boucle articulatoire stocke l'information verbale. Ces mémoires jouent le rôle de cache. Elle est plus rapide et facile d'accès que la direction centrale.

Mémoire à long terme Le processus direction centrale interagit également avec la mémoire à long terme. Celle-ci est composée de trois mémoires : épisodique, procédurale et sémantique. Toutes ces mémoires sont utilisées lors d'une tâche de compréhension de programmes.

Mémoire épisodique La mémoire épisodique stocke l'information sur les éléments et les événements qui font partie de l'histoire de la vie de l'ingénieur logiciel. Elle stocke les connaissances du domaine et les connaissances fonctionnelles apprises par l'ingénieur durant sa période d'apprentissage et la connaissance du programme acquise durant la phase de compréhension du programme.

10. Le test chromatique d'Ishihara est un test permettant de déceler les problèmes de différenciation des couleurs rouges et vertes (par exemple, le daltonisme) [Wik08].

Mémoire procédurale La mémoire procédurale enregistre les informations sur les compétences et les procédures pour effectuer des actions. Les connaissances de programmation et les stratégies mises en place pour la compréhension logicielle sont contenues dans cette mémoire.

Mémoire sémantique Pour finir, la mémoire sémantique stocke les informations relatives à la connaissance des concepts. Les apparences visuelles des prototypes et catégories d'objets comme les architectures, les conceptions et les implémentations de prototypes sont stockées dans cette mémoire. Les autres connaissances du domaine (celles acquises par la vue ou construites par la direction centrale) sont également stockées dans cette mémoire.

Cette théorie décrit donc l'activité de compréhension de programmes de l'image rétinienne générée par une représentation graphique d'un modèle de programme jusqu'à la direction centrale, qui effectue l'activité de compréhension.

CHAPITRE 5

Mesure de l'effort de compréhension

> La qualité n'est jamais un accident ; c'est toujours le résultat d'un effort intelligent.
>
> John Ruskin

La norme ISO/IEC 9126 définit les métriques de maintenabilité interne comme "utilisées pour prédire le niveau d'effort requis pour la modification d'un produit logiciel" [ISO00b]. L'effort effectué pour réaliser une tâche de maintenance est donc un indicateur significatif de la qualité d'un logiciel en terme de maintenabilité. Cette simplicité de réalisation est dépendante de l'effort requis par le mainteneur. Ce chapitre a pour objectif de présenter la théorie relative aux mesures de l'effort de ces développeurs/mainteneurs.

La Section 5.1 présente la notion de Mental WorkLoad dont l'évaluation est décrite en Section 5.2. En outre, l'outil utilisé à cette fin dans notre expérience, NASA-TLX, est expliqué en Section 5.3.

Ce Chapitre est fortement inspiré des travaux de Stanton [Sta05].

Sommaire

5.1 La notion de *Mental Workload* (MW)

Wilson et al. définissent la charge mentale comme la partie de la capacité de traitement ou des ressources réellement nécessaires pour répondre aux exigences d'un système [Cai07, p. 2]. La mesure de l'effort global d'un individu effectuant une tâche (de maintenance sur un programme, par exemple) est une mesure difficile à évaluer. En effet, il est délicat de lier les faits observés (mesure du rythme cardiaque, *feedback* de l'utilisateur, etc.) à l'effort demandé par la tâche. Toutefois, il est possible que ces observations ne soient pas toutes liées à la tâche proprement dite mais à d'autres effets (stress dû au fait d'être surveillé, manque de sommeil, raisons psychologiques, etc.). De ce fait, il est difficile voire impossible de mesurer la "charge mentale" (MW : *Mental Workload*) à l'aide d'une seule variable.

L'analyse de la MW a été réalisée dans de nombreux domaines, tels que l'aviation ou la conduite automobile. Son évaluation demande l'utilisation de plusieurs mesures : la mesure de la performance de la tâche primaire (la tâche principale à effectuer par le sujet), des mesures de performances des tâches secondaires (temps de réactions, tâches intégrées), mesures physiologiques (variation du rythme cardiaque, rythme cardiaque) et des mesures subjectives (SWAT, NASA TLX).

5.2 Évaluation de la charge mentale

- La **performance de la tâche primaire** inclut la mesure de la capacité de l'opérateur (du sujet) à effectuer une ou plusieurs tâches sous analyse. Il est supposé que sa performance varie de façon inversement proportionnelle à sa charge mentale [Sta05]. L'avantage de cette mesure est sa facilité de mise en œuvre étant donné que ces mesures sont de toute façon réalisées. Par contre, il est tout à fait possible qu'un opérateur pourrait effectuer une tâche de façon efficiente sous une haute MW.
- La **performance des tâches secondaires** inclut la mesure de la capacité de l'opérateur à effectuer une ou plusieurs tâches secondaires en plus de la tâche primaire (par exemple, les opérations arithmétiques, trouver un élément dans un schéma, se souvenir d'un résultat précédemment obtenu...). L'utilisation de ces mesures est basée sur l'hypothèse que plus la charge de travail de l'opérateur augmente, plus sa capacité à réaliser la où les tâches secondaires va diminuer. Cette mesure permet une meilleure granularité du résultat et permet d'évaluer la répartition de la MW mesurée de la tâche primaire dans les différentes tâches secondaires. Un désavantage de cette mesure est son manque de sensibilité dû aux variations mineures de la MW. Également, l'intrusion de cette mesure dans la performance de la tâche principale n'est pas à négliger.
- Les **mesures physiologiques** de la MW incluent tous les aspects physiques qui pourraient être affectés par une variation de la MW. Le rythme cardiaque et sa variation, les mouvements oculaires et l'activité du cerveau ont déjà été utilisés pour estimer la charge mentale. Ces mesures n'influent pas sur les mesures de la performance de la tâche principale et peuvent souvent être faites durant tout type d'expérience. Le sujet n'a également que peu d'emprise sur ces paramètres (dans le cas ou le sujet essayerait de biaiser le test). Cependant, celles-ci ont un coût élevé et peuvent obstruer les mouvements de l'opérateur, peuvent manquer de fiabilité et dans certains cas peuvent manquer de précision.
- Les **mesures subjectives** peuvent être effectuées durant ou après l'exécution de la tâche principale. Elles consistent en un *feedback* de l'opérateur où celui-ci évalue lui-même sa charge mentale. Les avantages principaux de ces mesures sont sa facilité et sa rapidité d'application, son

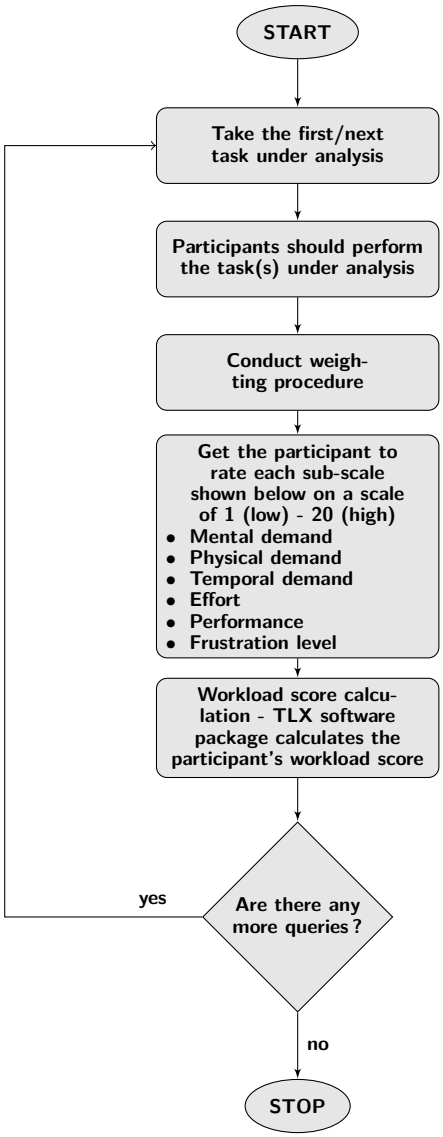


FIGURE 5.1 – Utilisation de TLX tiré de [Sta05]

faible coût et le fait que ces mesures soient non-intrusives vis-à-vis de la tâche primaire. Malgré ces avantages, ces mesures sont souvent semblables à celles de la performance de la tâche principale. Les participants ont également tendance à oublier certaines variations de leur MW en fonction de la réussite de la tâche principale ou de son échec.

5.3 NASA-TLX

TLX est un système de mesure subjectif créé par la NASA et utilisé pour calculer la charge mentale basée sur six sous-échelles : la demande mentale, la demande physique, la demande temporelle, l'effort, la performance et le niveau de frustration [NAS03].

Demande mentale Quelle demande mentale et quelle activité de perception a été requise (par exemple, penser, décider, calculer, se rappeler, regarder, rechercher...) ? Est-ce que la tâche était facile ou a demandé beaucoup d'efforts, simple ou complexe ?

Demande physique Quelle était le niveau d'activité physique requise (par exemple, pousser, tirer, tourner, contrôler, activer...) ? Est-ce que la tâche était facile ou complexe, lente ou rapide, calme

ou ardue, reposante ou laborieuse ?

Demande temporelle Quelle pression temporelle avez-vous ressenti durant la tâche en raison de la vitesse ou du rythme de celle-ci ? Est-ce-que le rythme était lent et tranquille ou rapide et frénétique ?

Effort Avez-vous dû travailler difficilement (mentalement et physiquement) pour atteindre votre niveau de performance ?

Performance Comment évaluez-vous votre travail ? Pensez-vous avoir accompli tous les buts de la tâche demandée ? Étiez-vous satisfait de votre performance lorsque vous avez effectué la tâche ?

Niveau de frustration Durant l'exécution de la tâche, avez-vous ressenti un sentiment d'insécurité, vous êtes vous senti découragé, irrité, stressé et embêté ou au contraire vous êtes-vous senti satisfait, heureux, relax et content de vous ?

Chaque sous-échelle est présentée au sujet soit durant l'exécution de la tâche, soit après l'accomplissement de la tâche. Le sujet doit estimer chaque paramètre sur une échelle qui va de faible (1) à élevé (20). NASA-TLX est la méthode d'évaluation subjective de la MW la plus répandue. Elle a été appliquée à de nombreux domaines : civil, militaire, l'aviation, la conduite, les opérations en chambre de contrôle de centrales nucléaires, etc. Il existe également quelques extensions ([Sta05, p. 312)]. Le mode opératoire de la méthode est illustré à la Figure 10.2.

Le MW est ensuite calculé sur base des résultats fournis par le sujet. Il est également possible de pondérer les différentes variables en fonction de l'importance que l'expérimentateur accorde à chacune d'elle. Par exemple, dans le cadre d'une activité de réflexion sur papier, il est probable que le poids associé à la demande physique soit faible voire nul. Ces poids sont fixés par l'expérimentateur. Il est ensuite possible de calculer la MW à l'aide de la formule suivante :

$$\mathrm{MW} = \mathrm{MD} \times \mathrm{MDW} + \mathrm{PD} \times \mathrm{PDW} + \mathrm{TD} \times \mathrm{TDW} + \mathrm{OP} \times \mathrm{OPW} + \mathrm{FR} \times \mathrm{FRW} + \mathrm{EF} \times \mathrm{EFW}$$

où : MD, PD, TD, OP, FR, EF sont les valeurs (en pourcent) fournies par l'utilisateur et où MDW, PDW, TDW, OPW, FRW, EFW les poids associés à chaque sous-échelle.
Ci-dessous sont présentés les avantages et désavantages de NASA-TLX tels que tirés de [Sta05].

NASA-TLX fournit une technique **simple** et **rapide** et **multi-dimensionnelle** pour estimer la charge de travail d'un sujet. TLX est probablement la **technique la plus utilisée** pour estimer la charge de travail d'un sujet [Sta05].

La décomposition en sous-échelles génériques permet une **application à n'importe quel domaine**. Dans le passé, TLX a été utilisé dans de nombreux domaines différents tels que l'aviation, le contrôle du trafic aérien, le nucléaire, la pétro-chimie et l'automobile. D'après Wierwille et Eggemeir [WE93], la technique TLX a montré qu'elle satisfaisait bien la demande des expérimentations en vol.

Cette utilisation a permis de s'assurer que NASA-TLX a été **soigneusement testé** dans le passé et a aussi été le sujet à de nombreuses **études de validation** telles que ce que proposent Hart et Staveland ([CCPE09]). Certaines études ont également montré sa supériorité sur SWAT [LG01, CCPE09].

NASA-TLX est fourni avec un paquetage logiciel qui permet d'enlever une charge de travail à l'expérimentateur, ce qui a pour conséquence de **simplifier et d'accélérer la procédure d'évaluation**. Ce paquetage est également disponible en version papier. Un autre avantage de TLX est le fait que

lorsque la tâche est évaluée après sa résolution, l'évaluation **n'influence en rien la résolution de la tâche principale**. Le fait d'effectuer l'évaluation TLX après la résolution de la tâche peut néanmoins introduire des **biais dans l'auto-évaluation du sujet**. En effet, le sujet peut avoir oublié, ou négligé certaines augmentations de sa MW. Cependant, si TLX est effectué durant l'exécution de la tâche principale, une influence sur celle-ci est possible.

Un autre inconvénient de TLX est le fait que **l'évaluation de la MW peut être corrélée à la performance de la tâche effectuée**. Par exemple, les sujets qui n'ont pas correctement effectué la tâche pourraient évaluer leur MW très haute et vice-versa. En outre, la **pondération des sous-échelles est laborieuse** et ajoute du temps à la procédure.

CHAPITRE 6

Processus expérimental en génie logiciel

> In theory, there is no difference between theory and practice. But, in practice, there is.
>
> Jan L. A. van de Snepscheut

EN génie logiciel expérimental, la conception d'expérience est primordiale. Pour répondre à cet objectif, la Section 6.1 introduit le domaine du génie logiciel expérimental, le met en contexte et en réalise un constat. Ensuite, les différentes stratégies utilisées en génie logiciel expérimental sont expliquées en Section 6.2 et sont accompagnées d'une description plus détaillée de la stratégie utilisée dans ce travail. Enfin, les principes de la procédure mise en pratique dans notre expérience sont décrits en Section 6.3.

La Section 6.1 se base notamment sur le livre de Pfleeger et al. [PA09] et de l'article de Basili [Bas93]. En outre, les Sections 6.2 et 6.3 sont majoritairement inspirées du livre de Wohlin et al. [WRH+99].

Sommaire

6.1 Génie logiciel expérimental

Écrire un logiciel est tout autant un art que la science et il est important de comprendre pourquoi. Les chercheurs en génie logiciel étudient les mécanismes de l'informatique et les théorisent pour les rendre plus efficients et productifs. Il existe plusieurs façons d'effectuer une tâche particulière sur un système particulier mais certaines d'entre elles sont meilleures que les autres. Une façon peut être plus efficiente, plus précise, plus facile à modifier, plus facile à utiliser ou plus facile à comprendre. Chaque programmeur est capable d'écrire un code fonctionnel, mais cela relève des compétences et de la compréhension d'un ingénieur logiciel professionnel de produire du code robuste, facile à comprendre et à maintenir et de rendre son travail le plus efficient possible. En conséquence, le génie logiciel concerne la conception et le développement de logiciels de haute qualité [PA09]. L'industrie du logiciel est fortement intéressée par le développement de logiciels d'une puissance industrielle et le domaine du génie logiciel se concentre sur comment construire de tels systèmes [Jal08, p. 2].

6.1.1 Définition

L'*Institute of Electrical and Electronics Engineers* (IEEE) propose une définition du Génie Logiciel dans son *Standard Computer Dictionary* [IEE91, p. 186] :

> "(1) L'application d'une approche systématique, disciplinée et quantifiable au développement, aux opérations et à la maintenance de logiciel ; c'est-à-dire l'application d'ingénierie au logiciel.
> (2) L'étude de l'approche décrite en (1). "

Le logiciel et le génie logiciel ont trois caractéristiques fondamentales [Bas93] :

1. Le logiciel peut être **complexe**, c'est-à-dire complexe à construire et à comprendre.
2. La première raison de cette complexité est le **manque de modèles**, spécialement des modèles flexibles de produits, de processus et toute autre forme de connaissance requise pour construire ou comprendre les solutions logicielles.
3. Le logiciel est créé via un **processus de développement**, pas par un processus de manufacture. Cela signifie véritablement que le logiciel est conçu par ingénierie.

6.1.2 Contexte

En tant qu'ingénieurs du logiciel, nous utilisons notre connaissance de l'informatique pour nous aider à résoudre des problèmes. Souvent, le problème avec lequel nous traitons est lié à un système informatique mais, parfois, les difficultés qui sous-tendent le problème n'ont aucun rapport avec l'informatique. Par conséquent, il est essentiel de comprendre en premier lieu les mécanismes ou les techniques de chaque problème que nous rencontrons. Nous devons résoudre le problème en premier. Ensuite, si besoin est, nous pouvons utiliser la technologie comme un outil afin d'implémenter notre solution.

La plupart des problèmes sont larges et parfois difficiles à gérer, spécialement s'ils représentent quelque chose qui n'a jamais été résolu auparavant. Il est donc nécessaire de commencer par étudier un problème en l'**analysant**, c'est-à-dire en le divisant en plusieurs morceaux plus compréhensibles et essayer de travailler avec eux. Nous pouvons donc décrire le problème plus large comme une collection de petits problèmes et de leurs inter-relations.

Une fois que le problème est analysé, nous devons construire notre solution à partir des composants qui s'adressent aux différents aspects du problème. La **synthèse** est l'assemblage d'une structure large à partir de plus petits blocs de construction. Afin de nous aider à résoudre un problème, nous employons une variété de méthodes, d'outils, de procédures et de paradigmes. Les ingénieurs du logiciel les utilisent pour améliorer la qualité de leurs produits logiciels. Leur but est d'utiliser des approches efficientes et productives afin de générer des solutions effectives à des problèmes [PA09].

6.1.3 Constat

Mais qu'en est-il de la réussite actuelle du génie logiciel ? Les logiciels permettent, à l'heure actuelle, d'effectuer nos tâches de traitement de texte, de calcul, de communication mais également d'assistance (médecine, agriculture, transport...) et bien d'autres choses. Néanmoins, le logiciel n'est pas sans problème. La plupart du temps, les systèmes fonctionnent, mais pas exactement comme espéré. En réalité, les fautes logicielles sont courantes et produire des logiciels exempts de bugs est difficile. Alors que certaines fautes sont plutôt ennuyeuses, d'autres coûtent véritablement du temps et de l'argent.

Par exemple, un projet (dans les années 1980) de traitement automatique de formulaires des impôts sur le revenu d'un montant prévu de 103 millions de dollars a nécessité 90 millions de dollars supplémentaires. Effectivement, le système s'est avéré inadéquat au vu de la charge de travail requise. De plus, ces problèmes ont empêché le remboursement des contribuables avant la date limite, ce qui a provoqué un surcoût de 40,2 millions de dollars d'intérêts et de 22,3 millions de dollars d'heures supplémentaires. Le problème n'était toujours pas résolu en 1996 [Saw85]. Ce cas n'est pas isolé au vu des études effectuées à propos des taux de réussite des projets informatiques [The95].

Des recherches sont nécessaires pour aider à l'établissement d'une base scientifique et d'ingénierie pour le génie logiciel. À cette fin, les méthodologies de recherche requièrent l'implication du besoin de construire, d'analyser et d'évaluer des modèles de processus et de produits logiciels [Bas93, p. 2].

6.1.4 Aspect transdisciplinaire

Dans les autres disciplines, il existe plusieurs exemples de méthodologies. Elles consistent en des formes variées de paradigmes expérimentaux ou analytiques. Les paradigmes expérimentaux requièrent une conception expérimentale, des observations, une collecte de données et une validation sur le processus ou le produit soumis à l'étude. Cette section décrit trois modèles expérimentaux. S'ils sont parfois similaires, ils tendent à mettre en évidence différentes choses. [Bas93, p. 2].

1. La **méthode scientifique** observe le monde, propose un modèle ou une théorie de comportement, mesure et analyse, valide les hypothèses du modèle ou de la théorie. Enfin, elle répète la procédure si possible.
   a) La **méthode d'ingénierie** observe des solutions existantes, propose de meilleures solutions, construit/développe, mesure et analyse. Enfin, elle répète le processus jusqu'à ce qu'aucune autre amélioration ne paraisse possible.
   b) La **méthode empirique** propose un modèle, développe des méthodes statistiques/qualitatives, les applique aux cas d'étude, mesure et analyse, valide le modèle et répète la procédure.
2. La **méthode mathématique** propose une théorie formelle ou un ensemble d'axiomes, développe une théorie, dérive des résultats et si possible, les compare avec des observations empiriques.

La méthode scientifique est utilisée dans les domaines appliqués, tels que la simulation d'un réseau de télécommunication pour évaluer ses performances, par exemple. La méthode d'ingénierie est probablement dominante dans l'industrie alors que les méthodes empiriques sont traditionnellement utilisées dans les sciences sociales et la psychologie [WRH+99, p. 5]. Ces paradigmes servent de base pour distinguer les activités de recherche des activités de développement. Si un de ces paradigmes n'est pas utilisé dans une étude, celle-ci n'est probablement pas un projet de recherche. Par exemple, la construction d'un système ou d'un outil seul est un développement et non de la recherche. La recherche implique le gain de compréhension à propos de comment et pourquoi un certain type d'outil pourrait être utile.

6.1.5 Paradigme expérimental en génie logiciel

Le domaine de l'Ingénierie du logiciel possède des théories qui sont d'une aide inestimable dans la mise en place d'expériences créées dans le but d'observer des faits prouvant (ou contredisant) les lois et les théories. Néanmoins, les théories existantes [ER03] n'expliquent pas les faits bien connus. Cela est dû à la relative jeunesse du domaine, à la complexité des phénomènes et au manque d'approbations et de reconnaissance des formalismes et des notations. Une théorie aide dans la compréhension d'un domaine en expliquant une liste de questions telles que "Pourquoi est-ce comme cela ?" [ER03, p. 7] [New73].

Les études empiriques sont essentielles pour comprendre le phénomène auquel les ingénieurs font face. Citons par exemple le travail de précurseurs tels que Basili [BRZ05]. En effet, cet auteur met en évidence la nécessité d'une approche plus scientifique dans les années 1980 [WRH+99, p. 5]. Les études empiriques sont basées sur le cycle classique : observation des faits, lois, théories. Plusieurs faits sont enregistrés au travers d'études empiriques et certaines lois sont proposées [Leh80]. Jusqu'ici, peu de théories ont été dérivées afin d'expliquer ces faits et ces lois et donc peu de nouveaux faits ont été prédits. Des chercheurs argumentent que les études empiriques actuelles se concentrent trop sur la généralisation et pas assez sur l'établissement de théories [JS04].

Brilliant et Knight définissent la recherche empirique comme :

> "*L*a recherche empirique peut être définie comme l'analyse basée sur l'observation d'une pratique réelle avec l'objectif de découvrir l'inconnu ou de tester une hypothèse. " [BK99]

Les études empiriques et les expériences en particulier sont importantes pour les chercheurs en génie logiciel. Les nouvelles méthodes, langages et outils ne doivent pas seulement être proposés, publiés et commercialisés. Il est crucial d'évaluer les nouvelles inventions et propositions en comparaison de celles existantes. L'expérimentation fournit cette opportunité et se doit d'en tirer parti [WRH+99, p. 4].

La raison principale d'utiliser l'expérimentation dans le génie logiciel est de rendre possible la compréhension et l'identification des relations entre différentes variables. Il existe nombre d'idées préconçues, mais ces idées sont-elles vraies ? Le paradigme orienté objet améliore t-il la réutilisation ? Ce type de questions peut être étudié dans le but d'améliorer notre compréhension du génie logiciel [WRH+99, p. 6].

6.1.6 Relation entre la recherche en logiciel et le développement en pratique

Une industrie doit comprendre le processus et le produit, définir leurs qualités, évaluer les réussites et les échecs, apporter un retour d'informations pour le contrôle des projets au travers de répétitions de processus fermés et apprendre des expériences de la recherche. Dans une telle situation, la recherche permettrait d'améliorer l'industrie, de rassembler les expériences réussies et d'en construire des compétences et de les réutiliser. Les techniques clés pour le support de tels besoins incluent la modélisation, les mesures, ainsi que la réutilisation de processus, de produits et d'autres formes de connaissance pertinente pour l'industrie en question. En conséquence, la recherche et les perspectives industrielles de l'ingénierie du logiciel ont une relation symbiotique [Bas93, p. 5].

6.1.7 Passé et perspectives

Boehm tente de retracer, pour chaque décennie depuis 1950, les principes et les pratiques considérées comme obsolètes ou intemporelles [Boe06]. Les principes intemporels qui ont amené à la conception actuelle du génie logiciel sont :

Ne pas négliger la science (1950) Il s'agit là de la première partie de la définition de l'ingénierie. Il ne faut pas se restreindre aux mathématiques et à l'informatique, mais inclure les sciences comportementales, l'économie, etc.

Réfléchir avant d'agir (1950) Les décisions prématurées peuvent être désastreuses.

Penser à contre-courant (1960) L'ingénierie répétitive n'auraient jamais créé l'Internet ou les interfaces utilisateurs.

Éliminer les erreurs le plus tôt possible (1970) La prévention des erreurs via une analyse de leurs causes est bénéfique.

Déterminer l'objectif du système (1970) Le paradigme But-Questions-Métriques en est un exemple.

Il existe plusieurs façons d'augmenter la productivité (1980) incluant l'équipe, l'entraînement, les outils, la réutilisation, l'amélioration du processus, le prototypage...

Ce qui est bon pour le produit est bon pour le processus (1980) incluant l'architecture, la réutilisabilité, la composabilité et l'adaptabilité.

Le temps est de l'argent (1990) Les investissements dans le logiciel ont généralement pour but d'obtenir un retour positif.

Rendre le logiciel utile pour les gens (1990) Il s'agit là de l'autre partie de la définition de l'ingénierie.

Considérer et satisfaire toutes les propositions de valeur des parties prenantes (2000) Si les parties prenantes cruciales sont négligées, elles contre-attaqueront généralement ou refuseront de participer.

6.2 Stratégies empiriques

Le génie logiciel expérimental dispose de plusieurs stratégies. Cette section a pour objectif de les expliquer brièvement et de présenter plus en détail celle utilisée dans notre approche, c'est-à-dire l'expérimentation.

6.2.1 Types de recherche

La **recherche qualitative** est concernée par l'étude des objets dans leur configuration naturelle. Un chercheur "qualitatif" tente d'interpréter un phénomène sur base des explications que les personnes lui apportent. Cette recherche part de l'acceptation du fait qu'il existe différentes méthodes d'interprétation. Elle est intéressée par la découverte de causes notifiées par les sujets de l'étude et par la compréhension de leur vue du problème manuellement. Le sujet est la personne qui prend part à l'expérience dans le but d'évaluer un objet [WRH+99, p. 7].

La **recherche quantitative** est principalement intéressée par la quantification d'une relation ou par la comparaison de deux groupes ou plus. L'objectif est l'identification d'une relation de cause à effet. Elles sont souvent conduites par le biais de la mise en place d'expériences contrôlées ou par la collecte de données dans des études de cas. Les études quantitatives sont appropriées lors de l'analyse de l'effet de manipulations ou d'activités [WRH+99, p. 7].

Il est possible pour les recherches quantitatives et qualitatives d'étudier les mêmes sujets, mais chacun d'entre-eux s'adressera à un type de question différent [WRH+99, p. 7].

6.2.2 Types de stratégie

En fonction du but de l'évaluation, s'il s'agit de techniques, de méthodes ou d'outils [1], et en fonction des conditions de l'étude empirique, il existe trois types principaux de stratégies expérimentales.

1. Une **enquête** est souvent une étude effectuée rétrospectivement lorsque, par exemple, un outil ou une technique est utilisé depuis un moment. Les premiers moyens de collecter des données qualitatives et quantitatives sont les questionnaires et les entrevues. Les résultats de l'enquête sont ensuite analysés pour dériver des conclusions descriptives et explicatives [WRH+99, p. 8].
2. Une **étude de cas** est utilisée pour la surveillance de projets, d'activités ou d'affectation. Les données sont collectées dans un but spécifique tout au long de l'étude. Basées sur la collection de données, les analyses statistiques peuvent être réalisées. L'étude de cas est normalement destinée à suivre un attribut spécifique ou à établir des relations entre les différents attributs [WRH+99, p. 8].
3. Une **expérience** est normalement effectuée dans un laboratoire, qui permet un haut niveau de contrôle sur les différentes variables pouvant avoir une influence sur le résultat étudié. Durant une expérience, des sujets sont assignés à différents traitements, et ce aléatoirement. L'objectif est de manipuler une ou plusieurs variables et de fixer toutes les autres variables à une valeur définie. L'effet de cette manipulation est mesuré et, sur cette base, une analyse statistique peut être effectuée [WRH+99, p. 9].

Tous les types de stratégies expérimentales peuvent être qualifiées de recherches qualitatives et quantitatives sauf l'expérience qui se révèle être une recherche qualitative.

6.2.3 Comparaison

Le Tableau 6.1 résume la comparaison des stratégies expérimentales décrites dans [WRH+99, p. 16]. Le code couleur de ce Tableau indique l'aspect favorable, ou non, d'un critère particulier pour une stratégie. La couleur rouge indique un critère défavorable, orange un critère indifférent et verte

1. Pour une définition de ces termes dans le domaine du logiciel, voir la définition de Pfleeger et al. [PA09] dans le glossaire en fin de document.

Facteur	Enquête	Étude de cas	Expérience
Contrôle d'exécution	Non	Non	Oui
Contrôle des mesures	Non	Oui	Oui
Coût d'investigation	Faible	Moyen	Élevé
Facilité de réplication	Élevé	Faible	Élevé

TABLE 6.1 – Comparaison des types de stratégies empiriques [WRH+99, p. 16]

un critère favorable.

Le *contrôle d'exécution* décrit le degré de contrôle qu'a le chercheur sur l'étude. Le *contrôle des mesures* est le degré avec lequel le chercheur peut décider quelles mesures seront collectées et inclues ou exclues durant l'exécution de l'étude. Le *coût d'investigation* est intimement lié au contrôle des mesures et peut être, par exemple, relatif à la taille de l'étude et des ressources nécessaires. La *réplication* a pour objectif de montrer que le résultat d'une expérience originale est valide pour une population plus large [WRH+99, p. 16].

6.3 Principe expérimental

En regard à la Section 6.2, la stratégie utilisée pour ce mémoire et décrite dans cette section est l'**expérience en laboratoire** selon les termes de Wohlin et al. [WRH+99].

6.3.1 Le processus expérimental

Les méthodes empiriques proposent un modèle et l'évaluent aux travers d'études empiriques telles que les expérimentations. Les études empiriques sont traditionnellement utilisées en sciences sociales et en psychologie, où il est impossible de tirer des lois de la nature, comme en physique. En sciences sociales et en psychologie, le facteur humain et son comportement jouent un rôle central. L'Ingénierie du logiciel est pleinement dirigée par le comportement humain au travers des personnes développant le logiciel. Il ne faut pas s'attendre à pouvoir tirer des règles ou des lois formelles du génie logiciel, si ce n'est, peut-être, si nous nous focalisons sur certains aspects techniques particuliers. Ce fait motive donc l'utilisation de l'expérimentation afin d'étudier l'influence de patrons architecturaux sur la maintenabilité des programmes étant donné que cette maintenance est effectuée par des humains, avec les connaissances, les réactions et les comportements qui leur sont propres. Le principe expérimental général est illustré en Figure 6.1.

Selon [WRH+99, p. 25], une mesure est un "*mapping*" de l'attribut d'une entité vers une valeur mesurable, typiquement une valeur numérique. Il existe deux types de variables dans une expérimentation, les variables dépendantes et les variables indépendantes. Les variables que nous souhaitons étudier sont appelées **variables dépendantes**. Typiquement, il n'existe qu'une seule variable dépendante dans une expérience. Dans un processus expérimental, toutes les variables qui sont manipulées et contrôlées sont appelées **variables indépendantes**. Une expérience étudie l'effet du changement d'une ou plusieurs de ces variables. Elles sont appelées **facteurs**. Les autres variables indépendantes sont contrôlées et fixées à une valeur définie durant l'expérience. Si ce n'était pas le cas, il serait impossible d'affirmer si c'est un facteur ou une autre variable qui cause les effets observés. Un **traitement** est une valeur particulière d'un facteur.

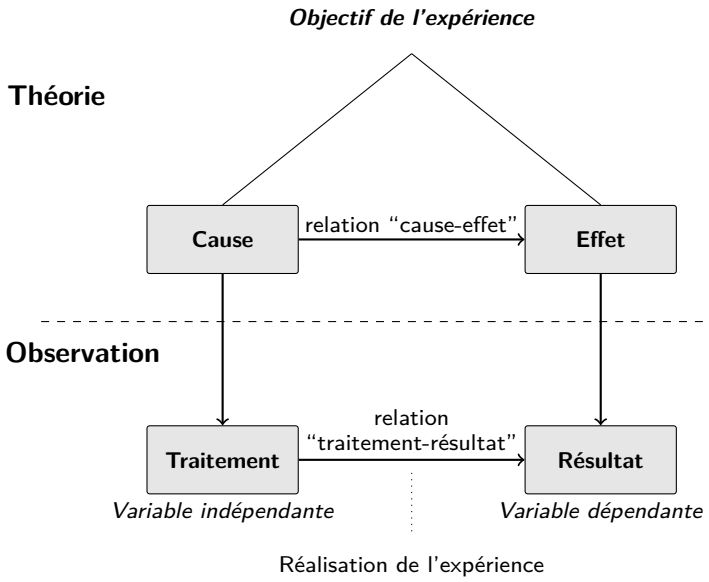


FIGURE 6.1 – Le principe expérimental [WRH+99, p. 32]

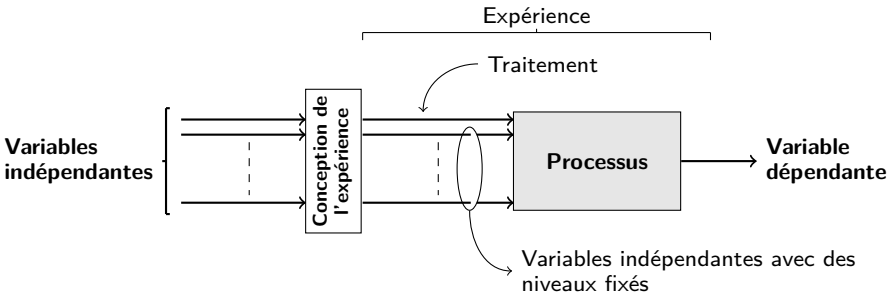


FIGURE 6.2 – Illustration d'une expérience [WRH+99, p. 34]

Les traitements sont appliqués sur une combinaison d'**objets** et de **sujets**. Un objet peut, par exemple, être un document ou un diagramme qui doit être corrigé à l'aide de certaines techniques. Les personnes qui appliquent ce traitement sont appelées sujets. Les caractéristiques des objets et des sujets peuvent être des variables indépendantes de l'expérience. La Figure 6.2 illustre ces éléments dans le processus expérimental.

Une expérience consiste en un ensemble de **tests** où chaque test est une combinaison d'un traitement, sujet et objet. Le nombre de tests affecte l'erreur expérimentale et fournit une opportunité d'estimer l'effet réel de tout facteur expérimental. L'erreur expérimentale nous aide à connaître quelle confiance il est possible de placer dans le résultat d'une expérience.

Le processus expérimental comporte plusieurs étapes, illustrées par la Figure 6.3, qui sont :

- Définition
- Planning
- Opération
- Analyse et interprétation
- Présentation et paquetage

Ces étapes sont détaillées dans les sections suivantes (hormis l'étape *présentation et paquetage* qui

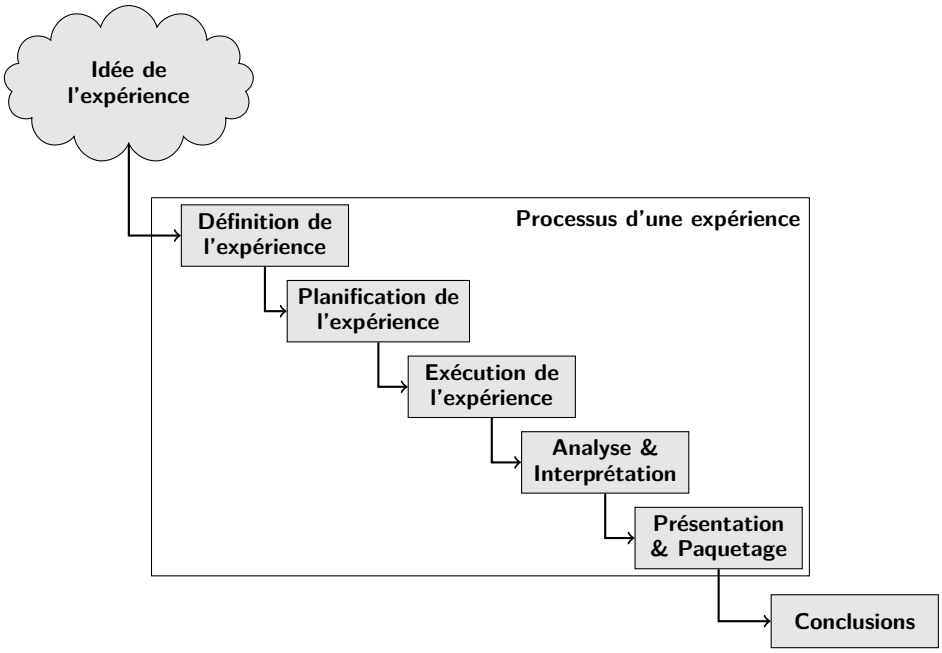


FIGURE 6.3 – Enchaînement des phases d'une expérience [WRH+99, p. 33]

s'avère triviale).

6.3.2 Définition

Conduire une expérience est une tâche demandant une grande quantité de travail. Afin de ne pas augmenter cette charge de travail, il est nécessaire de s'assurer que les buts de l'expérience peuvent être atteints. Dans cet objectif, il est indispensable de clairement énoncer l'expérience qui va être exécutée. À ce stade, l'hypothèse doit être formulée clairement, mais elle ne doit pas être établie formellement. À cette fin, Wohlin et al. [WRH+99, p. 42] suggèrent l'utilisation du gabarit suivant :

> "Analyze <Object(s) of study>
> for the purpose of <Purpose>
> with respect to their <Quality focus>
> from the point of view of the <Perspective>
> in the context of <Context> " [WRH+99]

Objet d'étude L'objet d'étude est l'entité qui est étudiée par l'expérience. Il peut être un produit, un processus, une ressource, un modèle, une métrique ou une théorie.

But Le but définit l'intention de l'expérience. Il peut être l'évaluation de l'impact de deux techniques différentes ou la caractérisation de la courbe d'apprentissage d'une organisation, par exemple.

Focalisation sur la qualité Il s'agit de l'effet principal sous étude dans l'expérience. Il peut être question de fiabilité, coût, etc.

Perspective La perspective donne le point de vue duquel les résultats de l'expérience sont interprétés.

Contexte : Le contexte est l'"environnement" dans lequel l'expérience est menée. Le contexte décrit brièvement quel personnel est impliqué dans l'expérience et quels artefacts logiciels sont utilisés dans cette expérience. Les sujets peuvent être caractérisés par expérience, taille d'équipe, charge de travail, etc. Les artefacts logiciels peuvent être caractérisés par taille, complexité, priorité, domaine d'application, etc.

		# Objets d'étude	
		Un seul	Plus d'un
# Sujets par objet	Un seul	Étude d'un seul objet	Étude de variation multi-objet
	Plus d'un	Étude multi-test dans l'objet	Étude bloquée sujet-objet

TABLE 6.2 – Caractérisation du contexte de l'expérience [WRH+99, p. 44]

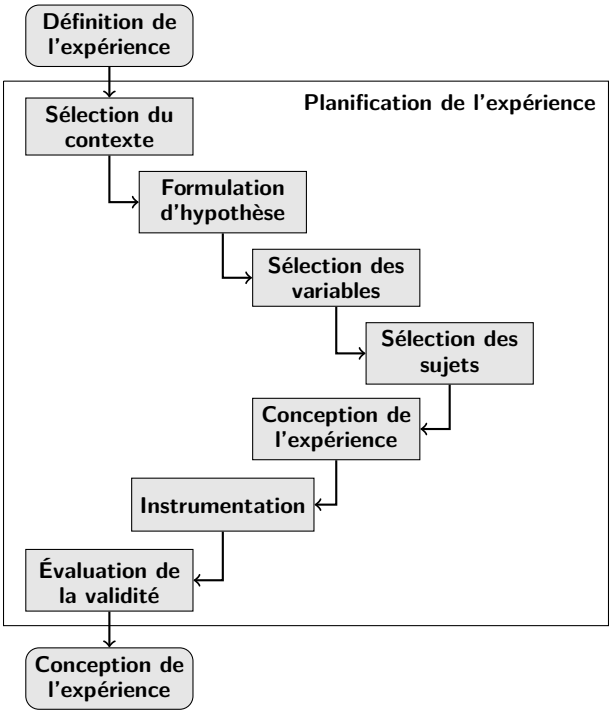


FIGURE 6.4 – Phase de planification d'une expérience [WRH+99, p. 48]

Le contexte d'expérience peut être caractérisé en terme du nombre de sujets et du nombre d'artefacts impliqués dans cette étude. Le Tableau 6.2 présente cette classification.

6.3.3 Planification

Une fois que la définition de l'expérience est posée et que nous avons déterminé *pourquoi* l'expérience est conçue, l'étape de planning permet de définir *comment* l'expérience est conduite. Cette étape est critique étant donné qu'elle va majoritairement influencer les résultats produits par l'expérience. De ce fait, ceux-ci peuvent être altérés voir détruits si l'expérience n'est pas planifiée correctement [WRH+99, p. 47].

L'étape de planification d'une expérience peut être divisée en six étapes tel qu'illustré en Figure 6.4. L'*input* de cette étape est la phase de définition de l'expérience comme décrit dans la section précédente. Sur base cette définition, l'étape de **sélection du contexte** sélectionne l'environnement dans lequel l'expérience sera exécutée. Ensuite, l'étape de la **formulation d'hypothèses** et celle de la **sélection des variables** dépendantes et indépendantes prennent place. La **sélection des sujets** est ensuite effectuée. La **conception de l'expérience** est choisie sur base des hypothèses et des variables sélectionnées. Ensuite, l'**instrumentation** prépare l'implémentation pratique de l'expérimentation. Finalement, l'**évaluation de la validité** vise à vérifier la validité de l'expérience.

Sélection du contexte

Afin d'obtenir des résultats les plus généraux possibles, une expérience devrait être réalisée sur des projets logiciels réels avec des sujets professionnels. Cependant, ce genre d'expérience est coûteuse et difficile à mettre en place. En effet, le fait d'effectuer une expérience sur un projet réel augmente le risque et peut ajouter du retard dans l'évolution du produit logiciel. Afin de réduire les risques et les coûts, il est parfois préférable de réaliser l'expérience sur des projets *off-line* avec des sujets étudiants [WRH+99, p. 48]. Ces projets sont une alternative plus économique et permettent un plus grand contrôle. Le contexte de l'expérience peut être caractérisé en fonction de quatre dimensions :

- Off-line VS On-line
- Étudiants VS Professionnels
- Jouets VS Réels Problèmes
- Spécifique VS Général

Formulation des hypothèses

La base pour l'analyse statistique d'une expérience est le test d'hypothèse. Une hypothèse est posée formellement et les données collectées durant l'exécution de l'expérience sont utilisées afin de, si possible, rejeter l'hypothèse. Si l'hypothèse peut être rejetée, des conclusions peuvent être tirées. Dans la phase de planning, la définition de l'expérience est formalisée en hypothèses. Celles-ci doivent être posées avant le début de l'expérimentation afin de pouvoir réaliser par la suite des tests statistiques sur les résultats. Deux hypothèses peuvent être formulées [WRH+99, p. 49] :

L'hypothèse nulle, H_0 Cette hypothèse affirme qu'il n'existe pas de réelles tendances ou patrons dans les paramètres de l'expérience ; les seules causes de différences proviennent des observations et sont des coïncidences. C'est cette hypothèse que l'expérimentateur veut rejeter avec la plus haute probabilité possible.

L'hypothèse alternative, H_a**,**H_1**,etc.** Cette hypothèse est en faveur de ce que l'hypothèse nulle a rejeté.

Le test d'hypothèse implique différents risques décrits ci-dessous :

Erreur de type I : Une erreur de type I apparaît lorsqu'un test statistique indique une relation alors que ce n'est pas le cas. L'erreur de type I peut s'exprimer comme suit :

$$P(\text{erreur de type I}) = P(\text{rejeter } H_0 | H_0 \text{ vrai})$$

Erreur de type II : Une erreur de ce type apparaît lorsque un test statistique n'indique aucune relation alors qu'il en existe une. Ce type d'erreur peut s'exprimer comme suit :

$$P(\text{erreur de type II}) = P(\text{ne pas rejeter } H_0 | H_0 \text{ faux})$$

Puissance : La puissance d'un test statistique est la probabilité que le test révèle une vraie relation si H_0 est faux. Un expérimentateur devrait choisir un test avec la plus haute puissance possible. La puissance peut être exprimée comme :

$$P(\text{Puissance}) = P(\text{rejeter } H_0 | H_0 \text{ faux}) = 1 - P(\text{erreur de type II})$$

Sélection des variables indépendantes

Le choix de telles variables requièrt généralement des connaissances du domaine. Les variables doivent avoir des effets sur la variable dépendante et doivent être contrôlables. Ce choix inclut également les échelles de mesure, la portée des variables et les niveaux spécifiques auxquels seront effectués les tests [WRH+99, p. 51].

Sélection des variables dépendantes

L'effet des traitements est mesuré dans la ou les variables dépendantes. Souvent, il existe seulement une seule variable dépendance et elle devrait par conséquent être dérivée directement de l'hypothèse. Elle n'est que rarement mesurable directement et nous devons plutôt la mesurer via une mesure indirecte. Cette mesure indirecte doit être attentivement validée car elle affecte le résultat de l'expérience. Le choix de variable dépendante signifie également que l'échelle de mesure et la portée des variables sont déterminées [WRH+99, p. 51].

Sélection des sujets

La sélection est intimement connectée à la généralisation des résultats de l'expérimentation. Afin de généraliser les résultats vers la population désirée, la sélection doit être représentative de cette population [WRH+99, p. 51-52].

Conception de l'expérience

Avant de concevoir l'expérience, il est nécessaire de développer toutes les idées au sujet des objectifs de l'expérience.

Une expérience consiste en des **séries de tests de traitements**. Pour exploiter au mieux l'expérience, les séries de tests doivent être attentivement planifiées et conçues. La conception de l'expérience décrit comment les tests sont organisés et comment ils s'exécutent. Pour concevoir l'expérience, nous devons étudier les hypothèses afin de voir quelles analyses statistiques doivent être réalisées pour rejeter l'hypothèse nulle.

Plusieurs aspects doivent être pris en compte au travers de **principes généraux** [WRH+99, p. 53-54] :

1. La **randomisation** est utilisée pour permettre à toutes les méthodes statistiques utilisées pour l'analyse des données d'être basées sur des variables indépendantes aléatoires. La randomisation s'applique à l'affectation des objets, des sujets et dans quel ordre les tests sont effectués. Elle est utilisée pour équilibrer l'effet de facteurs qui pourraient être présents. Elle est également utilisée pour sélectionner les sujets qui sont représentatifs de la population d'intérêt [WRH+99]. Son objectif est donc d'éliminer un biais dû à l'aspect systématique de l'expérience, une sélection arbitraire des sujets (biais de sélection), un biais accidentel ou une tricherie de l'expérimentateur [Bai08].
2. Parfois, un facteur peut avoir un effet sur les résultats sans que nous ne nous intéressions à cet effet. Le **blocage** peut être utilisé lorsque ce facteur est connu et contrôlable. Cette technique est employée pour éliminer l'effet indésiré de l'étude. Cela permet d'augmenter la précision de l'expérience. Ce principe n'étudie qu'un sous-ensemble (un "bloc") de situations au travers desquelles l'effet indésiré est toujours identique.
3. L'**équilibrage** permet d'assigner les traitements de telle façon qu'ils aient un nombre équivalent de sujets. Cette pratique est désirable car elle simplifie et renforce les analyses statistiques des données. Néanmoins, elle n'est pas indispensable.

Selon Wohlin et al., ces trois aspects doivent être négociés sur quatre types de conceptions standards. Selon le type de conception choisi, les types d'analyses et de méthodes applicables diffèrent.

Un facteur avec deux traitements Ces expériences désirent comparer deux traitements face à face. La méthode la plus commune est de comparer la moyenne des variables dépendantes pour chaque facteur.

Un facteur avec plus de deux traitements La comparaison est également réalisées à l'aide de moyennes. L'objectif d'une telle conception est identique à la précédente.

Deux facteurs Une telle expérience est plus complexe. La simple hypothèse pour les expériences avec un facteur seront divisées en trois hypothèses : une pour l'effet de l'un des facteurs, l'une pour l'autre facteur et une hypothèse pour l'interaction des deux facteurs. Nous notons τ_i l'effet du traitement i sur le facteur A, β_j l'effet du traitement j sur le facteur B et $(\tau\beta)_{ij}$ l'effet de l'interaction entre τ_i et β_j. Selon qu'un facteur est dépendant d'un autre ou non, la conception est respectivement imbriquée (conception imbriquée à deux étages) ou non (conception $2 * 2$). Le type d'analyse commun pour de telles conceptions est ANOVA.

Plus de deux facteurs La variable dépendante est influencée par un ensemble de facteurs et il est donc nécessaire d'analyser les interactions entre ces facteurs.

Instrumentation

Les instruments pour une expérience sont de trois types : les objets, les lignes de conduite et les instruments de mesure. Les **objets** peuvent être, par exemple, des spécifications ou un code source. Lors de la planification de l'expérience, il est important de choisir les objets qui lui sont appropriés. Les **lignes de conduite** sont nécessaires pour guider les sujets de l'expérience. Elles incluent, par exemple, des descriptions de processus et des *checklists*. Dans les expériences, les **mesures** sont guidées par la collecte de données à l'aide de formulaires, d'entrevues, etc.

L'objectif global de l'instrumentation est de fournir des méthodes de réalisation de l'expérience et de la surveiller, sans affecter le contrôle de l'expérience. Les résultats de l'expérience devraient être les mêmes indépendamment des choix d'instrumentation [WRH+99, p. 62-63].

Évaluation de la validité

Il est important de s'intéresser à la question de la validité dès la phase de planification dans le but de prévoir la validation adéquate des résultats de l'expérimentation. Une validité adéquate se réfère au fait que les résultats doivent être valides pour la population considérée. Premièrement, les résultats doivent être valides pour la population pour laquelle l'échantillon est sélectionné. Deuxièmement, il peut être intéressant de généraliser les résultats à une population plus large. Les résultats ont une validité adéquate s'ils sont valides pour la population que nous tenons à généraliser [WRH+99, p. 63].

Wohlin et al. [WRH+99, p. 64] présentent les quatre types de menaces à la validité introduits par Cook et Campbell.

1. La **validité interne** vérifie que le traitement implique les résultats. Si une relation est observée entre le traitement et les résultats, nous devons nous assurer qu'il s'agit bien d'une relation causale et qu'il ne s'agit pas du résultat d'un facteur sur lequel nous n'avons pas de contrôle ou qui n'est pas mesuré.
2. La **validité externe** est concernée par la généralisation. S'il existe une relation entre la construction de la cause et l'effet, le résultat de l'étude peut-il être généralisé hors de la portée de l'étude ? Existe-t-il une relation entre le traitement et les résultats ?

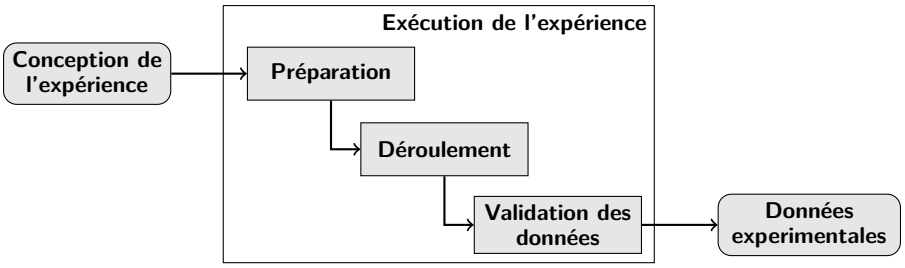


FIGURE 6.5 – Phase d'exécution d'une expérience [WRH+99, p. 76]

3. La **validité de construction** est intéressée par la relation entre la théorie et les observations. Si la relation entre la cause et l'effet est causal, nous devons nous assurer de deux choses : (1) que le traitement reflète la construction de la cause et (2) que le résultat reflète la construction de l'effet.
4. La **validité de conclusion** concerne la relation entre le traitement et les résultats. Nous voulons nous assurer qu'il existe une relation statistique significative.

6.3.4 Exécution

Durant la phrase d'exécution, les traitements sont appliqués aux sujets. Cette phase comprend trois étapes telles qu'illustrées en Figure 6.5 :

1. Les sujets sont choisis et formés durant la **préparation**.
2. Ils effectuent leurs tâches en fonction des différents traitements et les données sont collectées durant l'**exécution**.
3. Les données sont ensuite soumises à une phase de **validation**.

Préparation

Une meilleure préparation implique une plus grand facilité d'exécution de l'expérience.

Plusieurs aspects doivent être respectés quant au choix des participants. Il est tout d'abord impératif d'obtenir le **consentement** des sujets. Les participants doivent accepter les objectifs de recherche. S'ils ne connaissent pas l'intention du travail ou si le travail ne correspond pas à ce qu'ils en attendent, il existe un risque qu'ils n'effectuent pas les tâches selon les objectifs ou selon leurs capacités personnelles. Cela peut entraîner des résultats invalides. Il doit être clair pour les participants qu'ils sont libres de se retirer de l'expérience. Si les résultats obtenus peuvent être délicats pour les participants, il est important de leur assurer que les résultats sont **confidentiels**. Une façon d'attirer les gens dans une expérience est de leur offrir certains **avantages** sans pour autant tomber dans l'excès. En effet, un avantage trop élevé pourrait inciter les sujets à effectuer l'expérience uniquement par intérêt de l'avantage offert. **Tromper** les sujets est généralement à éviter. Si des méthodes alternatives sont disponibles, elles sont généralement préférables.

Avant le début de l'expérience, tous les instruments doivent être prêts et opérationnels. Dans la plupart des cas, il est plus approprié de préparer un ensemble personnel d'instruments pour chaque sujet.

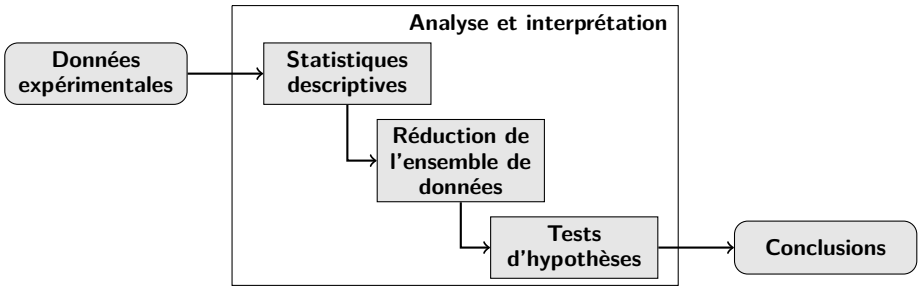


FIGURE 6.6 – Phase d'analyse et d'interprétation d'une expérience [WRH+99, p. 81]

Déroulement

L'expérience peut être effectuée de différentes manières. Alors que certaines expériences donnent lieu à un rassemblement de plusieurs sujets, d'autres nécessitent de prendre en compte les sujets un à un.

Les données peuvent être collectées par le sujet manuellement ou à l'aide d'un outil, au travers d'entrevues, ou automatiquement . La première est la plus commune alors que la dernière est la moins répandue. L'avantage des formulaires est le faible effort devant être fourni par l'expérimentateur. Cependant, les formulaires ne permettent pas de détecter directement les inconsistances, les incertitudes et les défauts du questionnaire. Les entrevues, elles, offrent la possibilité à l'expérimentateur de mieux communiquer avec les participants lors de la collecte de données. Néanmoins, elles demandent plus d'effort de la part de l'expérimentateur.

Si une expérience est effectuée dans un projet de développement classique, l'expérience ne doit pas affecter le projet plus que nécessaire. La raison est que l'expérience a pour objectif d'observer les effets de différents traitements dans un environnement tel que celui du projet. Si l'environnement du projet est trop modifié à cause de l'expérience, les effets peuvent être perdus [WRH+99, p. 78-79].

Validation des données

Lorsque les données ont été collectées, l'expérimentateur doit vérifier que les données sont raisonnables et qu'elles ont été collectées correctement. Par exemple, les participants peuvent avoir mal compris les formulaires et les avoir mal remplis. Une autre source d'erreurs est qu'un sujet n'ait pas effectué l'expérience sérieusement, il doit alors être écarté de l'analyse des données.

6.3.5 Analyse et interprétation

Les interprétations quantitatives peuvent être réalisées en trois étapes à partir des données collectées : les **statistiques descriptives**, la **réduction de l'ensemble des données** et les **tests d'hypothèses** (cf. Figure 6.6).

Statistiques inférentielles

Les statistiques négocient avec la présentation et le traitement numérique d'ensembles de données. Après une collecte de données, elles peuvent être utilisées pour décrire et présenter graphiquement les aspects intéressants de ces données. Le but des statistiques descriptives est d'avoir une idée de comment le jeu de données est distribué. Ces statistiques peuvent être employées avant la réalisation des tests d'hypothèses afin de mieux comprendre la nature des données et d'identifier les

données fausses ou anormales [WRH+99, p. 82].

La Section 7.2 s'adresse au lecteur intéressé par la compréhension des méthodes de statistiques utilisées dans notre approche.

Réduction de l'ensemble des données

Toutes les méthodes statistiques ont en commun que le résultat les utilisant dépendent surtout de la qualité des données en entrée. Si les données, sur lesquelles les méthodes statistiques sont appliquées, ne représentent pas ce que nous voulons qu'elles représentent, alors les conclusions des analyses ne sont pas correctes. Ces erreurs dans les données peuvent provenir d'erreurs systématiques ou peuvent apparaître comme marginales.

Il existe plusieurs façons de détecter des données marginales. Une méthode efficace, par exemple, est d'établir un diagramme de dispersion. La méthode utilisée doit être traitée au regard de la validité de l'expérience.

Lorsque les données marginales sont identifiées, il est nécessaire d'identifier leur cause. Si elles sont dues à un évènement rare ou étrange qui ne se produira plus jamais, elles peuvent être éliminées. Si cet évènement peut encore se produire, il est déconseillé d'exclure les données qui lui sont liées. Si ces données marginales sont dues à une variable qui n'a pas été considérée auparavant, les calculs et les modèles doivent en tenir compte.

Lors de l'analyse des données à l'aide de boîtes à moustaches par exemple, certaines données marginales nommées "*outliers*" sont détectées sans qu'il ne soit toujours possible d'en identifier la cause. De par cette incertitude, la question de leur suppression se pose lors de toute analyse statistique. Il existe, dans la littérature, un débat sur l'utilisation de ces valeurs extrêmes et de leur influence [OO04]. En l'occurence, **notre étude conserve les *outliers*** détectés par les méthodes statistiques car aucune raison à leurs valeurs extrêmes n'a pu être identifiée.

Les données invalides ne sont pas les seules à pouvoir être supprimées de l'ensemble de données. Il est parfois inefficace d'analyser des données redondantes si cette redondance est trop vaste [WRH+99, p. 90-92].

Tests d'hypothèses

Comme il l'est expliqué dans en Section 6.3.3, l'objectif des tests d'hypothèses [WRH+99, p. 92-96] est de voir s'il est possible de rejeter une hypothèse nulle H_0, basée sur échantillon d'une distribution statistique. Un cas habituel est que la distribution dépend généralement d'un seul paramètre. La mise en place de H_0 signifie formuler la distribution et l'assignation d'une valeur au paramètre qui sera testé.

Pour tester H_0, un cas de test t est défini et une zone critique C est donnée dans laquelle t est un cas qui rejette l'hypothèse de telle façon que :

- Si $t \in C$, alors H_0 doit être rejeté.
- Si $t \notin C$, alors H_0 n'est pas rejeté.

L'intention du test est de rejeter l'hypothèse.

Les tests peuvent être classifiés en deux catégories : les tests **paramétriques** et **non-paramétriques**.

1. Les tests *paramétriques* sont basés sur un modèle qui implique une distribution spécifique. Dans la plupart des cas, il est supposé que certains des paramètres impliqués dans un test paramétrique sont normalement distribués. Ces tests requièrent également que les paramètres soient mesurés sur une échelle d'intervalles. Dans le cas contraire, le test ne peut être utilisé.
2. Les tests *non-paramétriques* ne font pas le même type de supposition au sujet de la distribution des paramètres. Seules des suppositions très générales sont faites pour dériver des tests non paramétriques.

Peu importe la catégorie de tests considérée, deux facteurs sont à prendre en compte.

Applicabilité Il est important que les suppositions au sujet des distributions des paramètres et des échelles soient réalistes.

Puissance La puissance de tests paramétriques est généralement supérieure à celle des tests non-paramétriques.

Plusieurs types de tests existent et correspondent aux types d'expériences élaborées. La Section 7.2 détaille les tests utilisés dans ce document.

CHAPITRE 7

Éléments mathématiques

> Les propositions mathématiques sont reçues comme vraies parce que personne n'a intérêt qu'elles soient fausses.
>
> Montesquieu

DANS ce mémoire nous utilisons certains concepts mathématiques qu'il est nécessaire de définir formellement afin d'éviter les ambiguïtés. Ce Chapitre est destiné au lecteur intéressé par la clarification de certains concepts. Il n'a pas pour prétention d'être un receuil exhaustif et parfait des méthodes mathématiques employées, mais est uniquement utile pour décrire les outils mathématiques employés dans ce document.

Les Sections 7.1 et 7.2 décrivent respectivement les statistiques descriptives et inférentielles employées au travers de ce document. En outre, la Section 7.3 définit des concepts de distances utilisées dans notre étude.

Les définitions introduites en Section 7.3 sont traduites de l'ouvrage de Deza et Deza [DD09]. Les descriptions des tests d'hypothèses sont celles données par Wohlin et al. [WRH+99]. Le reste du chapitre est tiré du livre de Boslaugh et Watters [BW08].

Sommaire

7.1 Statistiques descriptives

7.1.1 Moyenne arithmétique

La moyenne d'une population est dénotée par la lettre grecque μ alors que la moyenne d'un échantillon est typiquement dénoté par une barre au-dessus du symbole de la variable [BW08, p. 55]. La moyenne $\bar{x}$ d'un ensemble de données x, exprimée sous une forme de sommation, est

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

où $\bar{x}$ est la moyenne de x, n est le nombre de cas et x_i est une valeur particulière de x.

Tout au long de ce document, le terme "moyenne" désigne exclusivement la moyenne arithmétique.

7.1.2 Quartiles

En statistiques descriptives, un quartile est chacune des trois valeurs qui divisent les données *triées* en quatre parts égales, de sorte que chaque partie représente un quart de l'échantillon de la population. Le 1^{er} quartile sépare les 25% inférieurs des données, le 2^e quartile est la médiane de la série et le 3e quartile sépare les 75% inférieurs des données. Pour une population de n valeurs, le 1^{er} quartile et le 3^e quartile sont respectivement la $Q1^{eme}$ et la $Q3^{eme}$ valeur de l'ensemble de données [Wik11i] où

$$Q1 = \lceil \frac{n}{4} \rceil$$

$$Q3 = \lceil \frac{n}{4} * 3 \rceil$$

7.1.3 Médiane

La médiane d'un ensemble de données (ou d'une population) est la valeur centrale lorsque les valeurs sont triées dans un ordre ascendant ou descendant. Si l'ensemble de données comporte n valeurs, la médiane est formellement définie par la $\frac{n+1}{2}$-ième valeur. Si n est une valeur paire, la médiane est la moyenne de la $\frac{n}{2}$-ième et de la $\frac{n+1}{2}$-ième valeur.

La médiane est une meilleure valeur de tendance centrale que la moyenne pour des données qui sont asymétriques ou qui contiennent des valeurs marginales [BW08, p. 57-58].

7.2 Statistiques inférentielles

Cette section présente différents tests d'hypothèses utilisés dans nos analyses de données. Alors que le *Mann-Whitney* est notamment utilisé pour l'analyse des facteurs confondants lorsqu'une seul facteur est considéré (avec deux traitements). ANOVA *Two-way* est la méthode la plus utilisée pour la majorité de nos tests d'hypothèse.

7.2.1 Test T

Le *t-test* est un test paramétrique utilisé pour comparer deux échantillons indépendants. La conception de l'expérience qui y est liée doit être *un facteur avec deux traitements*. Le t-test peut être réalisé sur base d'un certain nombre de suppositions [WRH+99, p. 99].

Les données exploitées par ce test sont deux échantillons indépendants $x_1, \ldots, x_n$ et $y_1, \ldots, y_m$ sur laquelle l'hypothèse nulle H_0 est généralement du type $\mu_x = \mu_y$ où μ_i est la moyenne d'un échantillon i.

Il est nécessaire de calculer S_p et t_0 qui en dépend. Dans ces deux définitions, S_x^2 et S_y^2 sont les variances individuelles des échantillons.

$$t_0 = \frac{\bar{x} - \bar{y}}{S_p \sqrt{\frac{1}{n} + \frac{1}{m}}}$$

$$S_p = \sqrt{\frac{(n-1)S_x^2 + (m-1)S_y^2}{n+m-2}}$$

Nous rejetons H_0 si $|t_0| > t_{\frac{\alpha}{2}, n+m-2}$. En l'occurrence, $t_{\alpha,f}$ est le pourcentage supérieur α de la distribution t avec le degré f de liberté [1], qui est égal à $n + m - 2$.

Le pendant non-paramétrique du *t-test* est le test de Mann-Whitney. C'est celui ci qui a été utilisé lors de nos analyses car une partie des données ne suit pas une distribution normale.

7.2.2 Mann-Whitney

Le test de Mann-Whitney est un test non-paramétrique alternatif au *t-test*. Il est toujours possible d'utiliser ce test a la place du *t-test* si les hypothèses faites par celui-ci semblent incertaines. Le test, qui est basé sur les rangs est résumé dans [WRH+99, p. 100]. Le lecteur intéressé pourra lire [SC88] ou [Ser88].

7.2.3 ANOVA

Le nom de cette méthode est "analyse de la variance" (*ANalysis Of VAriance*) car elle est basée sur l'observation de la variabilité totale des données et la partition de la variabilité selon différents composants. Dans sa forme la plus simple, le test compare la variabilité dûe au traitement et la variabilité due aux erreurs aléatoires [WRH+99, p. 105].

Analyse à un facteur

La forme la plus simple d'ANOVA est une analyse *One-Way*, où l'intention est de déterminer s'il existe un effet global de différents traitements d'une variable indépendante sur une variable dépendante. Le résultat de ANOVA est un *F-ratio*, qui peut être utilisé pour déterminer si des différences significatives existent entre les groupes.

L'hypothèse nulle est qu'une ou plusieurs populations diffèrent significativement par leurs moyennes.

Utiliser ANOVA nécessite de satisfaire plusieurs conditions. ANOVA suppose (1) l'indépendance des échantillons, (2) la normalité des distributions et (3) l'homoscédasticité (c'est-à-dire que les variances des distributions soient homogènes). L'hypothèse la plus importante, du point de vue de l'utilisateur, est l'égalité des variances.

La variation entre les moyennes est la base de l'analyse de la variance. Cette analyse est basée sur la moyenne totale – mesurée à partir de toutes les observations de tous les échantillons – et la moyenne

1. En statistiques le degré de liberté désigne le nombre de valeurs aléatoires qui ne peuvent être déterminées ou fixées par une équation [Wik10]

de chaque échantillon individuel. La variation totale au travers des différents échantillons est connu comme la **Somme totale des carrés** SS_{Total}, celle entre les échantillons est connue comme la **Somme des carrés des écarts inter-classes** $SS_{Between}$ (ou $SS_{Facteur}$) et la variation à l'intérieur des échantillons est la **Somme des carrés des écarts intra-classes** SS_{Within} (ou SS_{Residu}). La relation qui définit la somme totale des carrés des écarts est

$$SS_{Total} = SS_{Facteur} + SS_{Residu}$$

En d'autres termes, la *F-value* pour ANOVA est simplement le ratio entre $SS_{Between}$ et SS_{Within}. Par conséquent, si la variation entre les échantillons devient relativement grande comparée à la variation à l'intérieur des échantillons, il faut s'attendre à ce que les groupes soient tracés à partir de différentes populations et que la différence de moyenne soit significative [BW08, p. 232-239].

Analyse à deux facteurs

Cette analyse est utilisée lorsque deux facteurs sont en jeu et doivent être analysés. Comme la Section 6.3.3 le mentionne, il est nécessaire d'établir trois hypothèses nulles. Soient deux facteurs **A**, **B**, deux hypothèses H_{0a} et H_{0b} vérifient respectivement l'influence de **A** et de **B** sur la variable dépendante. Une troisième hypothèse H_{0ab} analyse l'impact conjoint des facteurs **A** et **B** sur la variable dépendante [BW08, p. 244-245].

Soient $SS_{Facteur_A}$ et $SS_{Facteur_B}$ les sommes des carrés des écarts inter-classes pour chaque facteur et $SS_{Interaction}$ la somme des carrés pour l'interaction des deux facteurs (qui est nulle en cas d'indépendance des facteurs), SS_{Total} est défini par

$$SS_{Total} = SS_{Facteur_A} + SS_{Facteur_B} + SS_{Interaction} + SS_{Residu}$$

7.2.4 Boîte à moustaches

La boîte à moustaches est une représentation graphique de données statistiques. Il s'agit d'une méthode compacte pour résumer et afficher la distribution d'un ensemble de données continues.

Ces graphiques sont toujours construits dans le but de mettre en évidence cinq caractéristiques de l'ensemble de données : la médiane, le premier et troisième quartile (et donc l'écart interquartile), le minimum et le maximum.

La tendance centrale, l'écart, la symétrie, et la présence de valeurs marginales dans un ensemble de données peut être vu un coup d'œil dans une boîte à moustaches et l'un à côté de l'autre, ces graphiques facilitent les comparaisons entre les différentes distributions de données [BW08, p. 71-73]. La Figure 7.1 est un exemple de boîte à moustaches pour les résultats d'élèves de 2009 à 2011. Le Tableau 7.1 présente les données du graphique. Il est donc facilement observable, par exemple, que l'année 2009 dispose principalement de meilleures notes que l'année 2010, mais avec une valeur maximale inférieure. L'année 2009 présente une valeur marginale (un *outlier*) de valeur 6 à l'aide du caractère ∘.

7.2.5 Valeur P

Une valeur p (ou *p-value* en anglais) exprime généralement la probabilité que les résultats au moins aussi extrêmes que ceux obtenus dans un échantillon soient dûs au hasard. Le terme "au moins aussi extrême" est nécessaire car la plupart des tests statistiques impliquent la comparaison

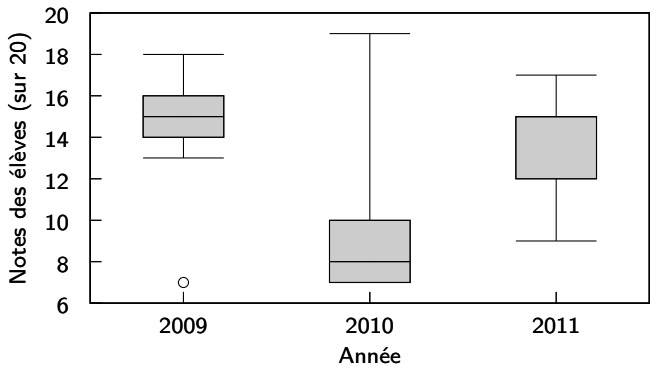


FIGURE 7.1 – Exemple de boîte à moustaches

	2009	2010	2011
1^{er} Quartile	14	7	12
3^{e} Quartile	16	10	15
Médiane	15	8	15
Minimum	13	7	9
Maximum	18	19	17

TABLE 7.1 – Données relatives à l'exemple de boîte à moustaches

du test statistique à des distributions hypothétiques. Les scores les plus proches du centre de la distribution sont les plus communs. Ces scores deviennent de moins en moins préférables lorsqu'ils s'éloignent du centre de la distribution [BW08, p. 145].

Lors d'un test d'hypothèse, la décision porte sur le rejet de l'hypothèse nulle et de l'acceptation de l'hypothèse alternative. Cette décision est liée à la recherche de résultats "significatifs". La valeur de la *p-value* de l'hypothèse soumise au test permet d'estimer si les résultats de l'échantillon sont suffisamment robustes pour rejeter l'hypothèse nulle. En pratique, il est habituel de considérer le rejet de l'hypothèse nulle si la *p-value* est inférieure à $\alpha = 0,05$. L'échec de rejet de l'hypothèse nulle n'est pas la preuve qu'elle est vraie mais seulement que l'étude n'est pas assez significative pour la rejeter [BW08, p. 142-143].

La *p-value* est donc la probabilité de commettre une erreur de première espèce (telle que décrite en Section 6.3.3).

7.3 Notions de distance

Deux types de distances sont utilisées dans ce document : la distance euclidienne et la distance de Levenshtein.

7.3.1 Distance euclidienne

La *distance* l_p notée d_{l_p}, où $1 \leq p \leq \infty$, est une métrique normée sur $\mathbb{R}^n$ (ou sur $\mathbf{C}^n$) définie par

$$||x - y||_p$$

où la $l_p - norme\ ||.||_p$ est définie telle que

$$||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}})$$

La **distance euclidienne** (ou la distance de Pythagore) d_E est une métrique sur $\mathbb{R}^n$, définie par

$$||x - y||_2 = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}$$

Il s'agit de la distance l_2 sur $\mathbb{R}^n$. L'espace de la métrique $(\mathbb{R}^n, d_E)$ est abrégé en $\mathbb{E}^n$ et est appelé l'*espace euclidien*. Dans un espace à deux dimensions, la distance euclidienne est une notion intuitive de distance entre deux points [DD09, p. 94].

7.3.2 Distance de Levenshtein

Considérons un alphabet comme un ensemble fini $\mathcal{A}$ où ses éléments sont appelés "caractères" ou "symboles" ($|\mathcal{A}| \geq 2$). Une chaîne de caractère (ou *string*) est une séquence de caractères sur un alphabet $\mathcal{A}$. L'ensemble de tous les *strings* est dénoté par $\mathcal{W}(\mathcal{A})$.

Une opération d'*édition* est une opération sur des *strings*. Sur un ensemble donné d'opérations d'édition $\mathcal{O} = \{\mathcal{O}_1 \dots \mathcal{O}_m\}$, la métrique d'édition correspondante entre deux chaînes de caractères x et y est le nombre minimum d'opérations d'édition de $\mathcal{O}$ nécessaires afin d'obtenir y à partir de x. Les opérations principales sont l'insertion, la suppression, le remplacement et l'échange.

Une permutation est toute chaîne de caractère $x_1 \dots x_n$ avec tous les x_i ayant une numéro différent dans $\{1, \dots, n\}$. $(Sym_n, ., id)$ est le groupe de toutes les permutations de l'ensemble $\{1, \dots, n\}$, où id est le *mapping* identité. La *métrique sur permutation de Hamming* d_H est une métrique d'édition sur Sym_n, obtenue pour $\mathcal{O}$ qui consiste au remplacement de caractères.

La **métrique de Levenshtein** (ou *edit distance*) est une métrique d'édition sur $\mathcal{W}(\mathcal{A})$, obtenue pour $\mathcal{O}$ qui consiste au remplacement, à la suppression et à l'ajout de caractères. La métrique de Levenshtein $d_L(x, y)$ entre deux chaînes de caractères $x = x_1 \dots x_m$ et $y = y_1 \dots y_n$ est égale à

$$\min\{d_H(x^*, y^*)\}$$

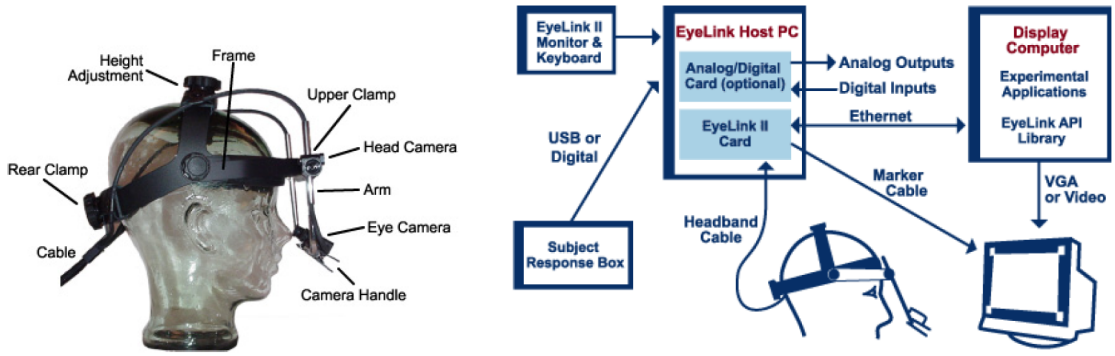
où x^*, y^* sont des *strings* de longueur k ($k \geq \max\{m, n\}$), sur l'alphabet $\mathcal{A}^* = \mathcal{A} \cup \{*\}$ de telle façon qu'après la suppression des nouveaux caractères $*$, les *strings* x^* et y^* soient respectivement réduites à x et y. Ici, l'*écart* est le symbole $*$. x^* et y^* sont des mélanges des *strings* x et y avec des *strings* qui consistent uniquement en des $*$. La **similarité de Levenshtein** est $1 - \frac{d_L(x,y)}{\max\{m,n\}}$ [DD09, p. 209-219].

CHAPITRE 8

Matériel

DIFFÉRENTS outils, qu'ils soient logiciels ou matériels, sont employés pour la collecte de données durant notre expérience et pour l'analyse de leurs résultats. La théorie de la Vision–Compréhension nécessite un oculomètre dont deux exemples sont décrits en Section 8.1. Ensuite, la Section 8.2 présente le logiciel que nous avons développé dans le but de traiter les données collectées à l'aide d'oculomètres. Davantage d'informations quant à ce logiciel sont disponibles dans les Annexes C et D ainsi que dans notre publication [DLG+11].

Sommaire



(a) Casque (b) Configuration

FIGURE 8.1 – Oculomètre Eye-link®II [SR 06]

8.1 Oculomètres

Dans cette section, nous présentons deux systèmes d'oculométrie de deux générations différentes : intrusifs (le sujet doit porter du matériel sur lui) et non-intrusifs.

Un oculomètre (ou *eye-tracker*) est un appareil permettant de connaître la position du regard d'un individu. Le fonctionnement et l'histoire de ces appareils peuvent être trouvés en Section 4.2.3. Nous présentons dans cette section les deux oculomètres utilisés au laboratoire *Ptidej*.

8.1.1 Eye-link®II

Le système Eye-link®II est composé de trois caméras miniatures montées sur un casque (illustré en Figure 8.1a). Deux caméras sont utilisées pour suivre le mouvement des yeux. La troisième caméra permet un suivi plus précis du point de vision du sujet. Il a été utilisé pour les expériences menées par Van Den Plas [Van09] et Jeanmart [Jea08].

La Figure 8.1b de *SR-Research*[1] [SR 06] décrit la configuration de ce système d'oculométrie qui utilise deux ordinateurs séparés : le PC Hôte et le PC d'affichage. Le PC Hôte effectue le suivi des yeux en temps réel à mesure de 250 à 500 mesures par seconde et calcule également la position réelle du regard du sujet sur l'écran. Ces données sont stockées dans un fichier sur le PC Hôte.

Le PC d'affichage, quant à lui, fournit l'image sujette à l'expérimentation et effectue les opérations de calibration. Les programmes d'affichage de données, le code source et les instructions pour créer un programme d'expérimentation sont fournis dans le *Eye-link®II Windows Developer Kit* mais le `Data Viewer`[2] n'est ni libre, ni à code ouvert (*open source*), ni gratuit et ne peut être étendu en aucune façon avec de nouvelles méthodes ou algorithmes d'analyse.

Le principal problème avec ce système est son instabilité. Si le sujet s'éloigne trop du point de calibration, l'étape de calibration doit être refaite. Un *offset* (voir Section 8.1.3) est présent pour chaque sujet dont on enregistre les mouvements oculaires.

La suite logicielle associée à cet appareil est capable de générer des fichiers de type `.edf`, format propriétaire qu'il est néanmoins possible de convertir en XML (via un programme nommé `edf2xml`).

1. `http://www.sr-research.com/accessories_ELII_dv.html`
2. `http://www.sr-research.com/accessories_EL1000_dv.html`

FIGURE 8.2 – Oculomètre FaceLAB [Fac]

Ce fichier comporte une suite d'éléments XML. Chaque élément de ce fichier comporte un temps de départ et un temps de fin (en millisecondes) et une durée (en millisecondes). De plus, un élément relatif à une fixation contient des coordonnées (x, y) et un élément relatif à une saccade contient des coordonnées de départ (x_1, y_1) et d'arrivée (x_2, y_2) ainsi que des informations concernant son amplitude et sa vitesse maximale.

8.1.2 FaceLAB and GazeTracker™

FaceLAB (de *Seeing Machines* [3]) est un système plus récent et beaucoup moins intrusif que l'Eye-link®II. En effet, le sujet n'a pas à porter de matériel particulier. Il est notamment utilisé dans une expérience menée par Cepeda Porras et al. [Por08]. Le système consiste en un ordinateur (portable ou fixe) auquel sont reliées deux caméras infra-rouges comme le montre la Figure 8.2. Les caméras enregistrent la position de la tête en utilisant comme points de repère les caractéristiques du visage de l'individu telles que les sourcils, les lèvres, le nez, etc. et utilisent ces données pour suivre et adapter automatiquement les coordonnées du regard du sujet.

Ce système fonctionne de pair avec le logiciel *GazeTracker™* (de *EyeResponse* [4]), qui est un système plus récent [Eye09] de visualisation de données que l'*Eye-link®II Data Viewer*, mais qui comme son concurrent n'est ni libre, ni à code ouvert, ni gratuit. Plusieurs oculomètres peuvent interagir les uns avec les autres. Une telle configuration est souvent utilisée dans les cockpits d'avions ou encore dans l'habitacle des voitures. Avec cette configuration, il est possible de suivre le regard d'un sujet même si celui-ci tourne fortement la tête. Il est également possible de lancer *FaceLAB* et GazeTracker™ sur le même ordinateur. Il n'est donc plus nécessaire de disposer de deux ordinateurs différents.

FaceLAB interagit avec l'oculomètre proprement dit et envoie les données à *GazeTracker™* par le réseau. *GazeTracker™* enregistre simplement les données en association avec l'image affichée à l'écran. Ce logiciel fournit une interface et des outils plus avancés que *Eye-link®II Data Viewer* mais ne peut pas être réellement étendu avec de nouvelles méthodes d'analyse ou de nouveaux algorithmes. Des vidéos de démonstration du logiciel ont été réalisées par nos soins afin d'aider les futurs utilisateurs du système. Ces vidéos sont disponibles en ligne [5].

Ce système permet de générer des fichiers contenant la liste de toutes les observations faites par l'appareil ainsi que quelques métriques calculées telles que le temps passé sur une image, le pourcentage de suivi (dans le cas où le sujet a quitté l'écran des yeux), le diamètre moyen de la pupille, etc. Les données sont produites sous la forme d'un fichier texte séparé par des virgules où chaque ligne correspond soit à une saccade ou à une fixation. Dans les deux cas, la ligne contient le nom de l'image relative à la question posée, une durée en secondes (avec une précision à la milliseconde), le

3. `http://www.seeingmachines.com/`
4. `http://www.eyeresponse.com/`
5. `http://www.ptidej.net/research/taupe/videos/`

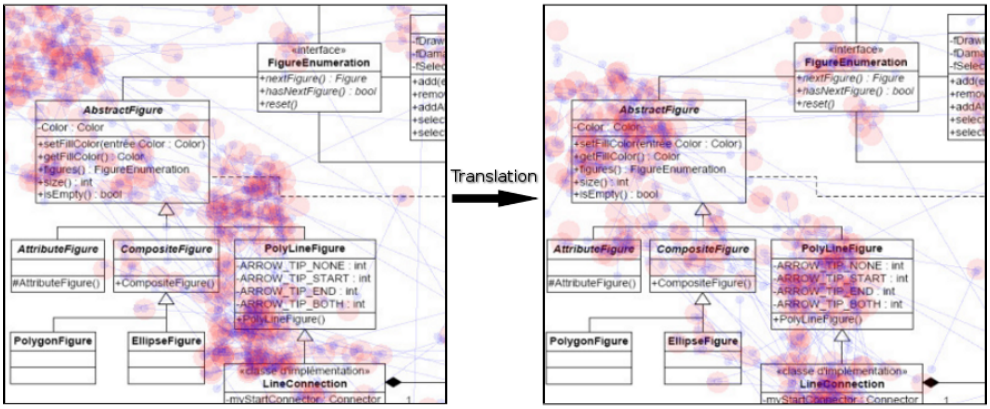


FIGURE 8.3 – Exemple d'*offset* statique

code indiquant s'il s'agit d'une saccade ou d'une fixation, deux coordonnées x et y, le diamètre de la pupille en largeur et en hauteur ainsi qu'un numéro d'identification. S'il s'agit d'une fixation, (x, y) correspond à sa position. S'il s'agit d'une saccade, (x, y) correspond au point de départ de la saccade (où le point d'arrivée sera la coordonnée de la saccade suivante).

8.1.3 La notion d'*offset*

Un des problèmes concernant les oculomètres est la présence d'*offset*. Un *offset* est un décalage, une différence entre l'endroit réel de la fixation et les coordonnées enregistrées par l'oculomètre. Nous avons classifié les *offsets* en trois catégories en fonction de leur nature. Ces trois types d'*offset* ont été rencontrés durant les expériences précédentes [Van09, Jea08] et la nôtre :

- Les ***offsets* statiques** sont des décalages constants entre la fixation enregistrée par l'oculomètre et ce décalage est constant quel que soit la zone de l'image concernée. Ils sont facilement identifiables et peuvent être corrigés en appliquant simplement une translation déterminée sur toutes les fixations comme illustré en Figure 8.3. Ce type d'offset est facilement identifiable visuellement étant donné que les "nuages" de fixations sont systématiquement décalés par rapport à l'image.
- Les ***offsets* non-statiques** sont des décalages constants mais différents en fonction de la zone de l'image concernée. Il est donc possible de déplacer des groupes de fixations par une translation mais celle-ci n'est pas applicable à tous les groupes de fixations. Ils sont moins facilement identifiables et les translations à effectuer pour les corriger sont différentes pour chaque groupes de fixations en fonction de leur position sur l'écran. Certains logiciels comme *GazeTracker* utilisent des algorithmes de correction. Ceux-ci utilisent les données fournies par l'utilisateur (qui déplace quelques points pour les accorder à l'image) pour interpoler le déplacement nécessaire pour tous les autres points et ce pour chaque zone de l'image.
- Les ***offsets* chaotiques** sont aléatoires et sont dûs à des problèmes liés à l'oculomètre. Par exemple, le sujet peut avoir fortement bougé la tête. Il n'est possible de corriger ces *offsets* que manuellement (voir Section 11.3.2).

8.2 Taupe

TAUPE : *Thoroughly Analyzing the Understanding of Programs through Eyesight* [Gué06] est un logiciel conçu par la *Ptidej Team* [6] pour importer des données de systèmes d'oculométrie et permettre l'exécu-

6. `http://www.ptidej.net`

	v0	v1	v2.0
Nombre total de lignes	3193	8036	55058
Nombre total de lignes de code	812	4912	12238
Nombre de Packages	3	11	25
Nombre de Classes	10	71	145
Nombre d'Interfaces	0	10	7
Nombre de Méthodes	40	490	800
Nombre d'Attributs	30	248	333
Manque de cohésion des méthodes	0.329	0.37	0.225

TABLE 8.1 – Évolution des métriques de TAUPE en fonction des versions

tion de divers algorithmes sur les données recueillies. TAUPE est actuellement en version (`2.0`) dont nous sommes les développeurs. Ci-dessous se trouve l'évolution du logiciel :

- La **version initiale** permettait simplement d'importer les données d'un seul système d'oculométrie (Eye-link®II) et proposait un système simple de visualisation des fixations et des saccades. Il a été écrit par Yann-Gaël Guéhéneuc.
- **TAUPE `1.0`** a été écrit et maintenu par plusieurs étudiants en stage (principalement par Bertrand Van Den Plas) au *Ptidej Lab*. La version `1.0` permettait à l'utilisateur d'appliquer plusieurs algorithmes sur les données, notamment un algorithme de correction d'*offset* statique (voir Section 8.1.3).
- Durant l'automne 2010, la totalité du logiciel a été ré-écrite par nos soins (**Version `2.0`**). Comme expliqué plus loin dans cette section, cette version est capable de gérer plusieurs types de fichiers générés par plusieurs systèmes d'oculométrie différents et contient de nouvelles fonctionnalités. Il s'agit de la version actuelle du logiciel. Son développement a duré plusieurs mois sous la supervision de Yann-Gaël Guéhéneuc.

Afin de donner une idée de l'évolution du logiciel, les métriques présentées en Tableau 8.1 ont été calculées à l'aide du *plugin* de Eclipse nommé Metrics [7].

La Figure 8.4 est une capture d'écran de l'interface principale de TAUPE. L'interface se divise en trois parties principales : la partie de gauche est le menu permettant de choisir la fonctionalité à exécuter (l'*AOI maker* : cf. Section 8.2.4, les résultats (exécutions d'algorithmes), l'outil de visualisation (cf. Section 8.2.4) et l'obtention du contenu du cache du logiciel (pour faciliter le débogage).

8.2.1 Motivation

TAUPE est initialement écrit pour comparer les différentes façons d'observer et d'analyser des diagrammes UML. Différentes questions sont posées au sujet de la compréhension de diagrammes et celui-ci doit effectuer un ensemble de tâches de maintenance sur ceux-ci. Les mouvements des yeux sont enregistrés en utilisant un oculomètre. Plusieurs logiciels existent dans le domaine de l'analyse des données et proviennent souvent des suites logicielles fournies avec les systèmes oculométriques, mais aucun n'est *open source* ou libre. De ce fait, il est difficile de personnaliser ces logiciels et d'y intégrer de nouvelles méthodes d'analyse des données collectées. Notre choix s'est donc porté sur la diffusion du logiciel TAUPE sous la licence *GNU GPL v3* [8].

7. `http://metrics.sourceforge.net/`
8. `http://www.gnu.org`

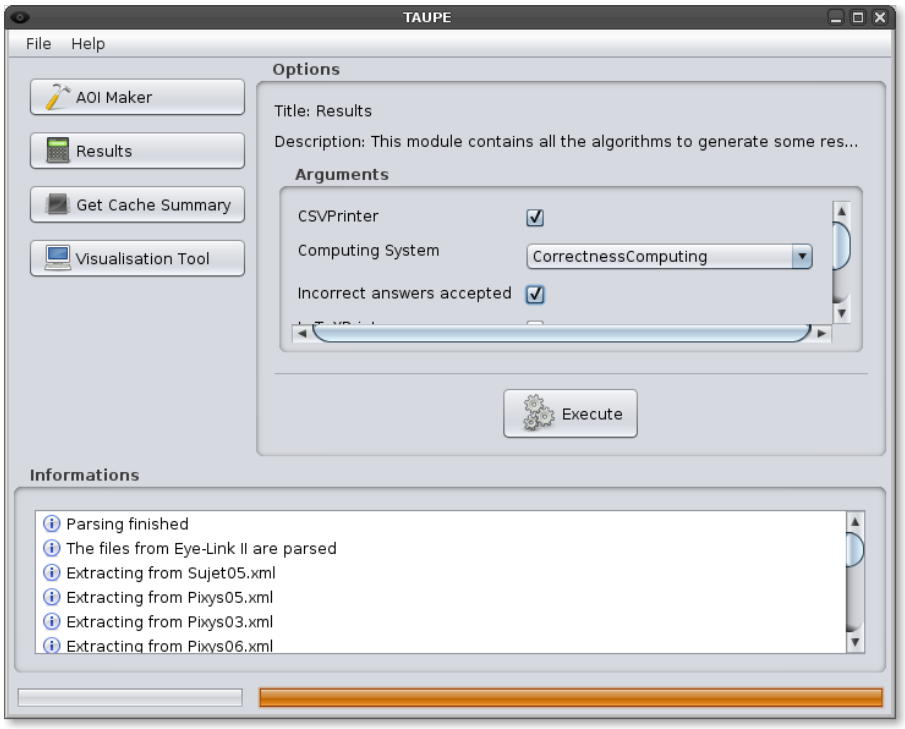


FIGURE 8.4 – TAUPE : interface principale

Les logiciels d'analyse de données fournis avec les systèmes d'oculométrie ne permettent que de générer des statistiques simples sur les fixations. Ces informations sont trop pauvres pour être utilisées dans un contexte scientifique. Deux logiciels étaient utilisés par l'équipe *Ptidej* : *Eye-link®II Data Viewer* (tel que décrit en Section 8.1.1) et le couple *FaceLAB/GazeTracker*^{TM} (cf. Section 8.1.2).

8.2.2 Architecture conceptuelle

Comme décrit dans la Figure 8.5, le composant principal de TAUPE est l'ensemble de données que sont les *fixations* et les *saccades*. Ces données sont fournies par les systèmes d'oculométrie. Comme expliqué en Section 8.1.1, certaines manipulations sur les données sont parfois nécessaires. Par exemple, les données générées par Eye-link®II doivent être converties en XML par l'outil `edf2xml`.

Les données dans TAUPE sont organisées en fonction des réponses aux questions fournies par les sujets. Les questions sont représentées par un ensemble d'images et leurs zones d'intérêt correspondantes. À chaque question correspond un fichier texte contenant la définition des zones d'intérêt liées à cette question. Ces fichiers peuvent être écrits manuellement mais TAUPE permet malgré tout à l'utilisateur de créer les fichiers de zones d'intérêt à l'aide d'une interface graphique `AOI Maker`. Les différents types d'analyseurs syntaxiques [9] (qui ont pour objectif de gérer les fichiers venant de différents oculomètres) sont utilisés pour extraire l'information des fichiers générés par les systèmes oculométriques.

Il est également possible de connecter un sujet à d'autres à l'aide de TAUPE via la notion de groupes. Un groupe est un ensemble de sujets qui ont un attribut en commun. Par exemple, cet attribut peut être le sexe, le niveau d'étude (Bac, Master, PhD...), ou encore le niveau de connaissance en UML (faible, moyen, élevé...). Toutes ces variables peuvent être évaluées par des moyens additionnels

9. Le terme *parser* est traduit dans ce document comme "analyseur syntaxique"

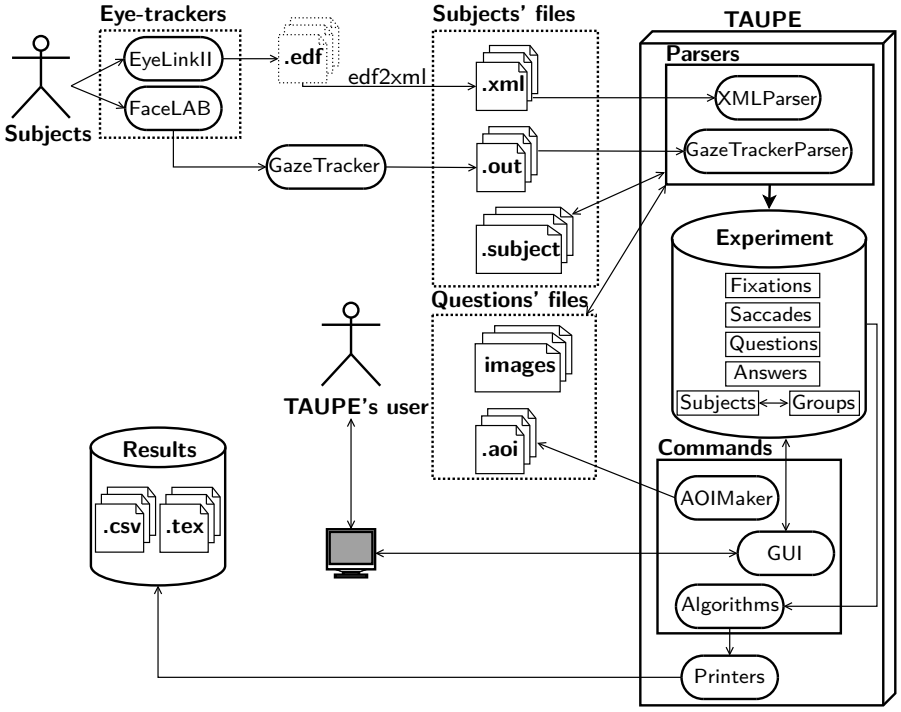


FIGURE 8.5 – TAUPE : architecture conceptuelle

tels que des questionnaires, des interviews... Le logiciel crée et gère ces groupes dynamiquement en fonction des caractéristiques des sujets.

Les principales fonctionnalités du logiciel sont appelées *commandes*. Par exemple, l'*AOI Maker* est l'une d'elle. La fonctionnalité la plus importante de TAUPE est la possibilité d'exécuter des algorithmes sur les données et de produire des résultats. Ces résultats peuvent être écrits dans des fichiers en utilisant des *printers* choisis par l'utilisateur. Actuellement, deux *printers* sont disponibles : LaTeX et CSV. Ces résultats peuvent être directement étudiés ou analysés dans un logiciel de *datamining*. En outre, TAUPE possède également une interface graphique permettant à l'utilisateur de visualiser les données et leurs caractéristiques.

8.2.3 Inspirations

Cette section présente les éléments et fonctionnalités implémentées dans TAUPE et décrit les domaines d'où proviennent ces fonctionnalités.

Du domaine de l'oculométrie

Deux fonctionnalités principales proviennent de l'utilisation des oculomètres. Premièrement, TAUPE doit implémenter des analyseurs syntaxiques afin de récupérer les données des fichiers générés par les oculomètres. Étant donné qu'il existe de nombreux systèmes d'oculométrie différents, il doit être facile d'ajouter de nouveaux analyseurs syntaxiques au programme. Deuxièmement, la contribution majeure de l'Eye-link®II est la notion d'*offset* (comme décrit en Section 8.1.3). Dans le but de tirer des conclusions en fonction de la position des fixations dans les zones d'intérêt, il est nécessaire de corriger ces fixations. Ce simple besoin a induit l'utilisation d'un logiciel externe capable de charger les données des oculomètres afin de corriger ces fixations.

Du domaine d'étude

Le domaine de l'oculométrie nous a inspiré un bon nombre de métriques permettant d'évaluer l'effort de parcours du sujet, par exemple. Les métriques basées sur les fixations sont les plus communes. Ces métriques sont détaillées en Section 4.2.3. Nous avons également imaginé de nouvelles métriques telles que celles décrites en Section ci-dessous.

Innovations apportées au domaine

Une nouvelle notion est introduite dans TAUPE : la notion de *parcours visuel*. Cette nouvelle métrique est définie plus en détail dans le Chapitre 9. [Jea08] suggère une métrique nommée le *Taux normalisé de fixations par zone d'intérêt*, cette métrique est également intégrée à TAUPE.

8.2.4 Utilisation

TAUPE est fourni avec une documentation complète tant pour son évolution que pour son développement. Les deux documents (guide de l'utilisateur et guide du développeur) peuvent être trouvés en Annexes C et D. La Figure 8.4 représente l'interface principale de TAUPE où l'accès aux différentes fonctionnalités est offert via un menu de *commandes*.

Zones d'intérêt dans TAUPE

Comme mentionné en Section 8.2.2, chaque question est liée à un fichier qui contient son ensemble de zone d'intérêts. Un utilisateur peut facilement créer ces fichiers en utilisant le module de TAUPE nommé "*AOIMaker*" (cf. Figure 8.6). Ces fichiers de zone d'intérêts peuvent également être écrits manuellement mais doivent respecter la grammaire suivante (décrite en *Extended Backus-Naur Form*). En outre, depuis sa dernière version, TAUPE ne considère plus qu'une zone d'intérêt est un rectangle mais bien un polygone. Une restriction quant aux polygones est qu'il doit s'agir uniquement de polygones de Jordan.

```
1 [<id> <type> <name> <coordonate> <coordonate> <coordonate>+
      <EOL>]*
2 <id> ::= integer
3 <type> ::= NULL | AOI | AORI
4 <coordonate> ::= "("integer "," integer")"
5 <EOL> ::= EndOfLine
```

La Figure 8.6 est une capture d'écran de l'interface de TAUPE permettant la création assistée des zones d'intérêts. Grâce à cet outil un utilisateur peut facilement générer les fichiers respectant les contraintes décrite dans cette section. Il est possible de spécifier l'identifiant, le nom et la pertinence de chaque zone.

Un outil de visualisation

La majorité des logiciels traitant des données provenant d'oculomètres permettent à l'utilisateur de visualiser la totalité des données enregistrées par les caméras. TAUPE introduit une innovation dans la visualisation (cf. Figure 8.7) en permettant à l'utilisateur de visualiser non seulement les données mais également les résultats de certains algorithmes. Par exemple, l'interface de visualisation est capable d'afficher l'enveloppe convexe pour un pourcentage de fixations données. L'utilisateur peut choisir le pourcentage de fixations à regrouper pour l'affichage. Le regroupement de fixations permet de regrouper deux fixations consécutives qui se trouvent dans une même zone. La durée de

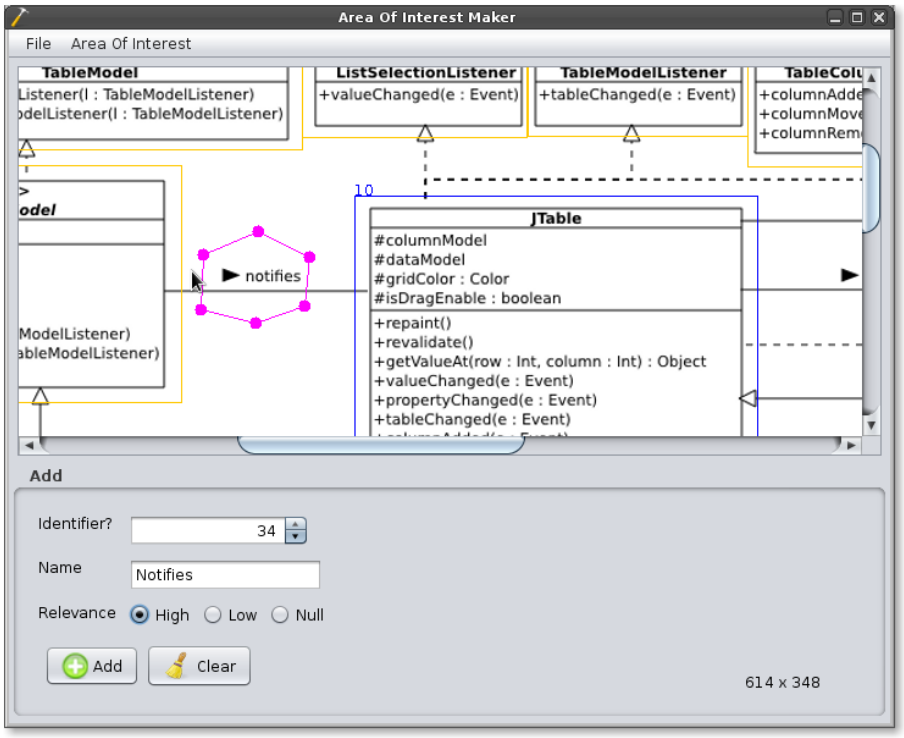


FIGURE 8.6 – TAUPE : interface graphique de création de zones d'intérêt

la fixation regroupée est la somme des durées des fixations qui la composent. Plus d'informations sur le logiciel se trouvent à l'adresse `http://www.ptidej.net/research/taupe/`.

La Figure 8.7 est une capture d'écran de l'interface qui permet de visualiser les fixations, saccades et certains résultats d'algorithmes. Une couleur peut être choisie pour chaque élément visualisable. Il est évidemment possible de choisir le sujet et la question dont on souhaite afficher les données.

8.2.5 Développement

En terme de développement, l'objectif principal de TAUPE est la modularité et la maintenabilité. Durant le développement, une attention particulière a été prêtée à ces deux aspects. Un simple développeur doit pouvoir facilement ajouter ses propres algorithmes, groupes et fonctionnalités. Ces objectifs étant bien entendu guidés par l'utilisation qui est faite du logiciel. Celui-ci étant utilisé par les scientifiques du laboratoire *PtiDej*, il doit pouvoir être facilement adapté aux besoins des différentes expériences.

Une question de maintenabilité

Le terme "maintenabilité" est ici utilisé tel que défini en Section 3.2. Afin d'assurer une certaine qualité en terme de maintenabilité du programme, il a été nécessaire d'écrire une documentation complète et détaillée. D'une part, des guides ont été réalisés (Annexes D et C) mais la *Javadoc* était également une priorité. Toutes les méthodes (802) possèdent leur *Javadoc*.

La maintenabilité de TAUPE est aussi liée à la facilité de maintenir un ensemble de fonctionnalités. Le guide du développeur inclut un ensemble d'explications destinées à guider le développeur pour la modification et l'ajout de différentes fonctionnalités. En résumé, les nouvelles fonctionnalités (telles que de nouveaux groupes, nouveaux algorithmes, nouveaux analyseurs syntaxiques, ...)

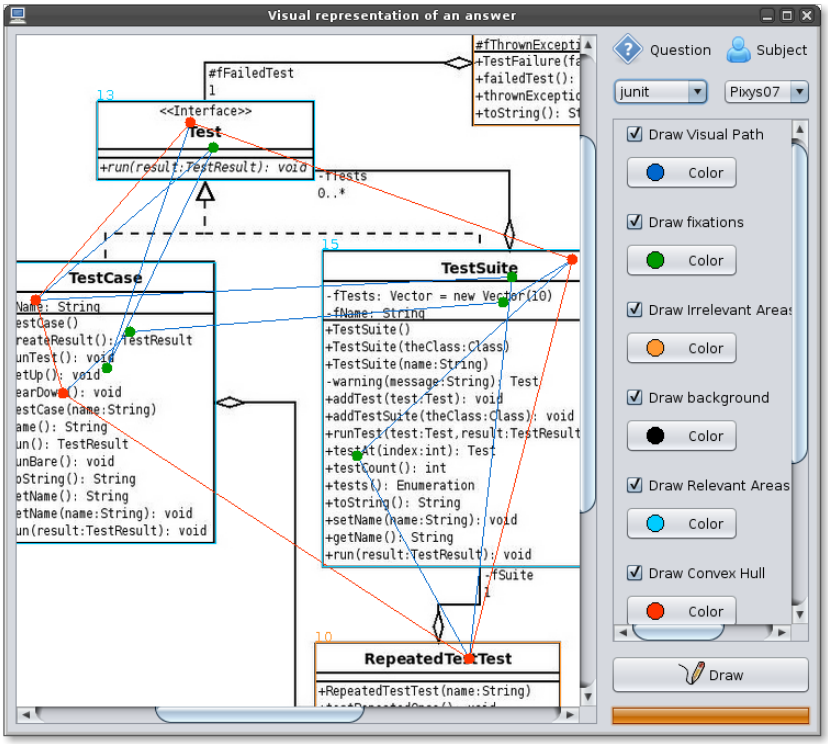


FIGURE 8.7 – TAUPE : outil de visualisation de données

peuvent être simplement ajoutés à TAUPE en étendant l'interface (ou la classe abstraite) correspondante. Il suffit ensuite de placer la classe dans le bon *package* Java et TAUPE est capable de charger cette classe dynamiquement sans autre modification dans le code. Le mécanisme qui permet ce chargement dynamique est le mécanisme de *reflection* Java. Par exemple, pour créer un nouveau groupe de sujets dans le système, le développeur doit uniquement créer une nouvelle classe qui étend la classe abstraite nommée `Group`. Il lui suffit ensuite de placer cette classe dans le *package* `laigle.data.viewer.utils.group.data`. Il existe cinq méthodes abstraites à implémenter pour rendre le groupe fonctionnel :

[getName() : String] Cette méthode retourne le nom du groupe en fonction de ses caractéristiques.

[getPrefixName() : String] Une version raccourcie de la chaîne de caractères de retour de la méthode `getName()`. Cette méthode est utilisée pour obtenir un nom court pour les fichiers de résultats. Par exemple, le sujet féminin numéro 7 aura comme nom "F7".

[getAvailableValues() : List<Object>] L'ensemble des valeurs possibles liées à la caractéristique commune des membres du groupe. Un groupe basé sur le sexe des individus, par exemple, aurait trois valeurs possibles : *homme*, *femme* ou *inconnu*.

[newInstance(Object o) : Group] Cette méthode joue le rôle de constructeur avec comme argument la caractéristique commune des individus du groupe.

[isEligible(s : SubjectData) : Boolean] Cette méthode vérifie si le sujet `s` peut appartenir au groupe ou non.

Tous ces éléments font que les nouvelles fonctionnalités et algorithmes peuvent être facilement ajoutés à TAUPE par les futurs chercheurs.

Choix d'implémentation

Depuis le début de la conceptualisation de l'architecture du logiciel, nous avons orienté le développement pour l'utilisation de certains patrons. En ce qui concerne l'interface graphique, c'est le patron architectural Modèle-Vue-Contrôleur (MVC) (selon sa définition de [GHJV94]) qui a été utilisé. Le module qui contient les algorithmes, les *printers* et les structures de résultats utilisent deux types de patrons : le patron structurel Composite et le patron de comportement Visiteur.

Les différents types de résultats utilisent une hiérarchie implémentant le patron Composite. Les *printers*, qui définissent comment produire les fichiers de sortie contenant les résultats, utilisent des méthodes `accept` tandis que les résultats décrivent comment parcourir leur propre hiérarchie (en utilisant des méthodes `visit`).

Étant donné que le logiciel ne gère qu'une expérience à la fois, la classe nommée `Experiment` est un Singleton. Le patron comportemental Itérateur est la méthode la plus utilisée pour parcourir les listes et les ensembles dans TAUPE. Le mécanisme de *reflection* de Java est largement utilisé dans le logiciel pour augmenter sa maintenabilité.

Validation et Vérification

TAUPE étant utilisé comme outil dans la recherche scientifique, il est crucial de s'assurer que tous les résultats fournis par le logiciel soient corrects. Dans ce but, TAUPE définit plusieurs tests unitaires utilisant le framework JUnit [10]. Le package `laigle.data.test` contient tous les cas de tests.

Afin d'améliorer la qualité du logiciel telle que définie dans la Section 3.1, nous avons mis en place une suite complète de tests unitaires. Ceux-ci assurent une meilleure maintenabilité de TAUPE. En effet, si un développeur souhaite introduire de nouvelles fonctionnalités ou de nouvelles structures de données, il lui suffit de lancer la série de tests unitaires afin de s'assurer qu'aucun "effet de bord" n'est apparu et que toutes les fonctionnalités précédemment implémentées sont toujours fonctionnelles et exemptes de bugs.

10. `http://www.junit.org`

CHAPITRE 9

Distance Relative entre Chemins Visuels

Ce chapitre présente une nouvelle métrique nommée "distance relative entre chemins visuels" (DRCV) qui permet une différenciation mathématique relative de deux parcours visuels.

Le contexte et la nécessité de son invention sont détaillés en Section 9.1, la métrique est ensuite définie dans la Section 9.2. Afin d'illustrer sa viabilité, une étude de cas basée sur les données collectées par Van den Plas [Van09] est présentée en Section 9.3. Enfin, une conclusion sur cette métrique est énoncée en Section 9.4.

Sommaire

9.1 Contexte

Lors de l'observation des parcours visuels des sujets durant l'observation des diagrammes, nous remarquons plusieurs phases :

1. Le sujet parcours rapidement la totalité des classes (survol du nom des classes).
2. Le sujet effectue un deuxième passage sur les classes qu'il a jugées pertinentes pour la tâche, c'est-à-dire une lecture rapide des méthodes des classes retenues à l'étape 1.
3. Le sujet lit en détail les classes qu'il a estimées intéressantes.

Néanmoins, aucune métrique n'existe à l'heure actuelle pour quantifier la différence entre deux parcours visuels considérés comme des séquences de zones parcourues. Par conséquent, cette section présente une tentative de création de métrique afin de mesurer cette différence.

9.2 Définition de la métrique

L'idée principale de cette métrique est de comparer des parcours visuels deux-à-deux et de déduire une métrique qui quantifie la différence entre ces deux parcours. Plus la valeur de cette métrique est élevée, plus la différence entre deux parcours visuels est grande.

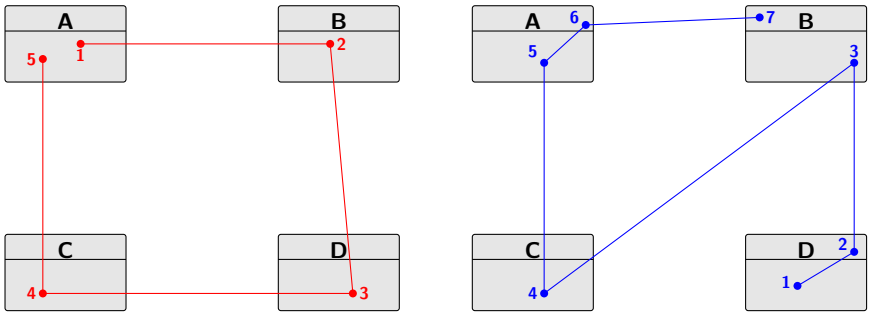
9.2.1 Représentation

Plusieurs métriques tentent déjà de représenter le parcours visuel mais aucune, à notre connaissance, ne considère la "dynamique" du parcours. Notre approche tient compte du parcours visuel comme d'une séquence chronologiquement ordonnée d'éléments visionnés par un sujet.

Le premier problème qui se pose est de déterminer le type d'éléments qui doivent être considérés comme "noeuds" [1]. En effet, un certain niveau de granularité doit être adopté par la métrique. Alors qu'il serait possible de considérer qu'un seul pixel soit un élément pertinent, il est également possible de considérer des zones de taille fixe qui découpent l'écran en différentes parties. La pertinence de ces éléments détermine la pertinence du parcours visuel considéré. L'état de l'art de la théorie de la vision identifie déjà des éléments importants que sont les zones d'intérêt (cf. Section 4.2.3). Ces zones seront celles considérées par la métrique. Nous définissons donc un parcours visuel comme une suite d'éléments parcourus chronologiquement par le regard d'un sujet. La Figure 9.1 illustre deux parcours visuels différents pour un même diagramme de classes UML où chaque classe correspond à une zone d'intérêt.

La deuxième réflexion doit se porter sur la visualisation multiple d'une même zone d'intérêt. Si une zone d'intérêt est visitée deux fois mais que le parcours visuel est passé par une autre zone d'intérêt entre temps, alors les deux passages dans la même zone sont considérés comme différents. Si ces deux passages dans une même zone sont successifs, alors ils sont considérés comme identiques. Par exemple, le parcours visuel de la Figure 9.1a passe deux fois par la zone d'intérêt **A** : en (1) et en (5). Ces deux passages par la zone **A** sont considérés différents. Par contre, le parcours de la Figure 9.1b passe successivement par **A** en (5) et en (6). Ces deux passages par **A** ne sont considérés que comme un seul passage. Ce choix se justifie de la façon suivante : un sujet qui maintient son regard dans une zone sans la quitter s'y intéresse une fois, peu importe le nombre de fixations qui correspondent à

1. Le terme de "noeud" provient initialement de la théorie des graphes. En effet, notre première représentation du parcours visuel se basait sur cette théorie.



(a) ABDCA (b) DBCAB

FIGURE 9.1 – Deux parcours visuels différents

cet intérêt[2].

Ensuite, que faut-il considérer comme une "visite", c'est-à-dire comme un "passage" dans une zone d'intérêt ? La donnée la plus fiable dont nous disposions à l'heure actuelle est la fixation. Une zone d'intérêt est donc considérée comme visitée si elle contient une fixation. La liste des fixations triée chronologiquement représente le parcours visuel "brut" du sujet. Le parcours visuel que nous traitons, lui, est constitué de zones d'intérêt et non pas de fixations.

Enfin, le parcours visuel doit disposer d'une représentation sous forme de données afin de pouvoir être traitée par un algorithme. Deux choix ont été considérés. La première est une représentation sous forme de chaîne de caractères où chaque zone d'intérêt est identifiée par un caractère. Par exemple, le parcours visuel de la Figure 9.1a est `ABDCA` tandis que celui de la Figure 9.1b est `DBCAB`. Cette représentation pose un problème majeur qui est le nombre de caractères disponibles. En effet, le nombre de zones d'intérêt représentables est restreint avec une telle approche. La solution se situe dans notre deuxième représentation où chaque zone d'intérêt est identifiée par un objet (en terme de programmation orientée objet). Les deux solutions ont leur intérêt. Effectivement, la première solution a un aspect plus théorique qui sera utilisé dans la suite de ce chapitre et la deuxième solution a la vocation d'être implémentée dans le logiciel TAUPE décrit en Section 8.2.3.

9.2.2 Différenciation

La représentation du parcours visuel n'est pas anodine. Effectivement, il existe une famille d'algorithmes qui permet de calculer la *distance* entre deux chaînes de caractères (*string*), nommée *edit distance* ou distance de Levenshtein. Alors que la distance de Levenshtein est formellement définie en Section 7.3.2, la section suivante présente une approche pratique et une implémentation à cette notion.

Distance de Levenshtein

L'*edit distance* entre deux chaînes de caractères se concentre sur la transformation d'un *string* en un autre par une série d'opérations d'édition sur des caractères individuels. Les opérations permises sont [Gus97, p. 215] :

- L'**insertion** I d'un caractère dans le premier *string*.

2. En l'occurrence, répéter plusieurs fois une même zone d'intérêt dans la séquence (en fonction du nombre de fixations par exemple) n'est pas envisageable. En effet, la métrique utilisée ensuite sur cette séquence serait totalement biaisée. Sans entrer dans le détail, un algorithme de *plus longue sous-séquence commune* (LCS) [CS75] serait plus adapté. Cette éventualité est un travail futur potentiel.

Algorithme 1 : Algorithme de calcul de l'*edit distance* entre deux chaînes de caractères

Entrées :
`s: char[1..m]` la première chaîne de caractères où `m` ≥ 0
`t: char[1..n]` la première chaîne de caractères où `n` ≥ 0
Résultat : La valeur ($\in \mathbb{N}^+$) de l'*edit distance* de `s` et `t`

```
d ← new integer[0..m,0..n]
for i ← 0 to m do
    d[i,0] ← i;
end
for j ← 0 to n do
    d[0,j] ← j;
end
for j ← 1 to n do
    for i ← 1 to m do
        if s[i] = t[j] then
            d[i,j] ← d[i-1,j-1];
        else
            d[i,j] ← minimum(d[i-1,j]+1,d[i,j-1]+1,d[i-1,j-1]+1);
        end
    end
end
return d[m,n];
```

- La **suppression** S d'un caractère dans le premier *string*.
- La **substitution** (ou remplacement) R d'un caractère dans le premier *string* par un caractère du second.

Gusfield définit :

> "*L'edit distance* entre deux chaînes de caractères est définie telle que le minimum d'opérations d'édition – insertion, suppression ou substitution – nécessaires pour transformer la première chaîne de caractères en la seconde. " [Gus97]

L'*edit distance* est parfois référée comme *distance de Levenshtein*. Sa définition implique que toutes les opérations sont faites sur une seule chaîne de caractères. Néanmoins, l'*edit distance* est parfois considérée comme le nombre minimum d'opérations faites sur les deux chaînes pour transformer les deux en une troisième chaîne commune. Cette vision est équivalente à celle définie ci-dessus, car une insertion dans un *string* peut être vue comme la suppression dans l'autre et vice-versa [Gus97, p. 216].

Algorithme initial

En tenant compte de la première représentation du parcours visuel décrite en Section 9.2.1, c'est-à-dire la chaîne de caractères, l'application d'un algorithme permettant le calcul de la distance de Levenshtein entre deux parcours visuels est triviale. L'algorithme permettant de calculer cette distance est détaillé en pseudo-code dans l'Algorithme 1 tel que présenté par Wagner et Fischer [WF74, Wik11e]. C'est le résultat de cet algorithme appliqué à notre représentation du parcours visuel que nous avons nommé **distance relative entre chemins visuels**.

Algorithme modifié

Pour adapter l'algorithme de Levenshtein à la représentation sous forme d'objets, il faut tout d'abord modifier les arguments de l'algorithme. Les deux chaînes de caractères sont remplacées par

deux listes d'objets de type "Zone d'intérêt". La deuxième et dernière étape est d'établir une relation d'équivalence entre ces objets. Cette relation est simple à déterminer, il s'agit simplement de comparer l'identifiant de deux zones d'intérêt, identifiants qui sont préalablement définis. Le logiciel TAUPE intègre cet algorithme modifié en laissant la possibilité à l'utilisateur de choisir le pourcentage de fixations à considérer afin de ne garder que les fixations le plus pertinentes. Ce pourcentage fait référence à l'ensemble des fixations les plus longues. À la suite d'une analyse préliminaire, nos parcours visuels ne tiennent compte que des 60% des fixations les plus longues.

9.2.3 Alternative

Selon Gusfield, calculer la *similarité* est équivalent à calculer la distance de deux chaînes de caractères. Cette similarité est décrite dans [Gus97, p. 226]. Néanmoins, les circonstances de notre stage ont fait que cette métrique, souvent préférée à l'*edit distance*, n'a été entrevue que bien après l'adoption de la distance de Levenshtein.

9.3 Étude de cas

Cette section présente rapidement l'application de la métrique sur un cas concret afin d'illustrer son utilité.

9.3.1 Description de l'expérience

Tel que décrit en Section 4.2.3, Van den Plas [Van09] a étudié l'impact du patron de conception Composite et Observateur sur des tâches de compréhension et de modification de programme sur des diagrammes UML sur trois projets *open-source* : JUnit, QuickUML et ArgoUML.

Quatre métriques ont été sélectionnées comme variables indépendantes dans cette étude pour évaluer l'effort du sujet : la densité spatiale, la matrice de transition, la durée moyenne des fixations et le *ON target ALL* (voir Section 4.2.3). Le niveau d'expertise des sujets a été évalué en fonction de leur emploi. Un sous-ensemble des sujets étaient des experts de la société Pyxis [3] alors que les autres sujets étaient des étudiants du laboratoire *Ptidej* et du *Soccer Lab*.

Un des objectifs de cette expérience est de montrer si les experts parcourent les diagrammes d'une façon différente des débutants. Cependant, il n'a pas été possible de tirer de conclusions basées sur la simple étude statique des fixations proposée par Van den Plas. La nouvelle métrique qu'est la distance relative entre chemins visuels est donc appliquée sur les données récoltées par Van den Plas afin de déterminer si la façon de parcourir un diagramme est différente selon le niveau de connaissance du sujet.

9.3.2 Application de la métrique

Avant d'appliquer la nouvelle métrique aux données, nous posons l'hypothèse suivante :

- H_0 : Les valeurs permettant d'estimer la différence entre les parcours visuels des individus parcourant les diagrammes UML suivent la même distribution quel que soit le niveau d'expertise du sujet.

Le Tableau 9.1 présente les résultats de l'application de la métrique sur les données de l'expérience [Van09]. Ces données sont directement calculées à l'aide de l'outil TAUPE (cf. Section 8.2).

3. `http://www.pyxis-tech.com/en/home`

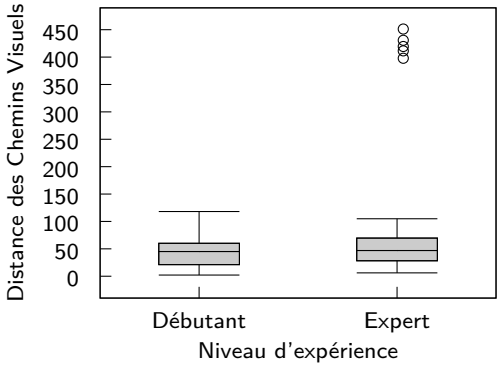


FIGURE 9.2 – Boîte à moustaches de la dispersion de la DRCV entre étudiants et professionnels pour les données de [Van09]

W	P-value
2301.5	0.02485

TABLE 9.1 – Test d'hypothèse Mann-Whitney des différences entre chemins visuels pour les données récoltées par Van den Plas [Van09]

9.3.3 Analyse des résultats

L'application de ce nouvel algorithme aux données recueillies par [Van09] met en évidence une tendance assez nette concernant le parcours visuel. En effet, il est mis en exergue que les sujets dits "experts" (provenant d'une société de développement) ont, en moyenne, un parcours visuel "plus différents" entre eux que les étudiants (ceux-ci ayant un parcours visuel plus semblable entre eux, une distance relative entre chemins visuels plus faible).

Nous avons donc ici affaire à une expérience mettant en avant un seul facteur (le niveau d'expérience des sujets) à deux traitements (expert et débutant) sur des données qui ne suivent pas une distribution normale. Afin de tester l'hypothèse H_0, nous appliquons donc le test d'hypothèse Mann-Whitney suivant les principes décrits dans [WRH+99, p. 97]. Le Tableau 9.1 présente les résultats obtenus.

Le test Mann-Whitney nous permet d'obtenir une *p-value* de 0.02485 ce qui est en dessous du seuil $\alpha = 0.05$. Il est donc possible de rejeter l'hypothèse H_0. En conclusion, il existe bien une différence dans la façon de parcourir les diagrammes UML en fonction du niveau d'expertise du sujet. Nous pouvons affirmer que les débutants semblent parcourir les diagrammes de façon plus systématique (leur moyenne de distance relative étant significativement plus faible) que les experts, ceux-ci utilisant leur expertise afin de déterminer plus rapidement les informations pertinentes d'un diagramme [DLG+11].

9.4 Conclusion et limites

L'analyse de la dynamique du parcours d'un diagramme de classes peut s'avérer précieuse pour mieux comprendre la façon dont les ingénieurs logiciels analysent un diagramme lors de la phase de compréhension de programmes.

La métrique **distance relative entre chemins visuels** peut être utilisée afin d'estimer et de quantifier la différence entre deux parcours visuels et peut ouvrir la voie à de nouvelles métriques se concentrant sur la dynamique du parcours et non plus seulement sur son aspect statique. L'application de la métrique sur les données de [Van09] a montré qu'elle peut s'avérer utile dans certains cas, mais pourrait sans nul doute être améliorée à l'avenir (cf. Section 16.2).

Cependant, cette mesure étant relativement simple, elle peut être considérée comme trop simpliste pour estimer la différence entre deux parcours visuels. En outre, le fait que la mesure soit relative et non absolue peut être vu comme une faiblesse. Enfin, se baser sur un algorithme d'*edit distance* particulier nous contraint aux défauts qui lui sont inhérents.

Deuxième partie

Étude empirique

CHAPITRE 10

Éléments constitutifs

L'idée de l'expérience ne remplace nullement l'expérience.

Emile Chartier

AVANT de détailler la conception de notre expérience, il est nécessaire de décrire tous les éléments qui la constituent. Les programmes étudiés, les patrons architecturaux considérés, la représentation visuelle utilisée et les sujets participants sont respectivement énoncés et justifiés en Sections 10.1, 10.2, 10.3 et 10.4. En outre, la Section 10.5 décrit la méthode utilisée pour appliquer les patrons dans les différents projets afin d'obtenir des diagrammes utilisables par les sujets dans le cadre de l'expérience. La Section 10.6 expose les choix que nous avons faits en terme de support matériel et d'environnement pour l'expérience. Enfin, les questions posées aux différents sujets sont expliquées en Sections 10.7 et 10.8.

Sommaire

10.1 Logiciels soumis à l'étude

Les tâches de maintenance que les sujets doivent effectuer s'exécutent sur des programmes ou des parties de programmes. Afin de déterminer les différents programmes à soumettre à l'étude, il est préalablement nécessaire de déterminer certaines contraintes qu'ils doivent respecter. Une fois les projets à analyser choisis au vu des différentes contraintes, nous devons les "réduire" afin d'en conserver un sous-ensemble étudiable. Ces programmes peuvent tout autant être des logiciels complets que des *frameworks*.

10.1.1 Contraintes

Chaque partie de programme soumise à l'étude (que nous nommerons PPSE dans la suite de cette section) se doit de respecter les sept contraintes suivantes.

C1 Le logiciel ou le *framework* lié au PPSE est **largement utilisé** par l'industrie et la communauté scientifique. Effectivement, notre étude vise à analyser des cas réels et non des projets isolés dans un objectif de réalisme.

C2 Le PPSE est écrit dans un **langage unique**.

C3 Le langage dans lequel est écrit le PPSE est un langage du **paradigme orienté objet**. De plus, aucun mécanisme complexe n'y est ajouté tels que l'inversion de contrôle ou le paradigme orienté aspect. Les mécanismes orientés objet présentés dans le livre de Liskov et Guttag [LG00] sont les seuls admis dans ces PPSE.

C4 Le PPSE **dispose d'un patron architectural** relatif à une interface utilisateur décrit en Section 2.

C5 La réduction du PPSE comporte **au maximum 40 classes** sans perdre son essence qu'est l'utilisation d'un patron architectural dans le cas d'une interface utilisateur.

C6 Le logiciel ou le *framework* lié au PPSE n'est pas un projet couramment étudié par les sujets. Les sujets qui participent à notre étude étant principalement des doctorants en génie logiciel expérimental, ceux-ci ont déjà effectué plusieurs études du même type.

C7 Le **code** du logiciel ou du *framework* lié au PPSE est **ouvert** afin de pouvoir en prendre connaissance.

10.1.2 Choix

Selon ces critères, nombre de logiciels utilisés dans l'industrie sont valides. Néanmoins, beaucoup d'autres ne le sont pas. Par exemple, le célèbre *framework* MVC *Struts2* ne respecte pas la contrainte **C2**. En effet, *Struts2* propose l'utilisation du XML afin d'écrire le composant contrôleur. Le logiciel IDE *Eclipse* s'avère respecter toutes les contraintes sauf **C5**. Effectivement, le fonctionnement de l'interface utilisateur de ce logiciel contient un trop grand nombre de classes et il n'est pas possible d'en réduire significativement le nombre sans perdre l'information essentielle à sa compréhension et à sa maintenance. Un logiciel tel que *JHotDraw* s'avère relativement intéressant pour notre étude. Cependant, les sujets l'ont rencontré dans plusieurs études, ce qui ne respecte pas la contrainte **C6**. Moins formellement, il est favorable de ne pas choisir des projets qui opèrent sur le même domaine d'application.

Le nombre de logiciels ou de *frameworks* à soumettre à l'étude est un paramètre important de l'expérience. Plus le nombre de logiciels/*frameworks* étudiés est faible, plus l'analyse de l'effet des patrons architecturaux est ternie par ces *frameworks*/logiciels. Néanmoins, plus ce nombre est élevé, plus

l'effet de fatigue des sujets doit être pris en compte car la durée de l'expérience en est directement dépendante. Notre choix s'est donc porté sur **deux** *frameworks* à étudier.

Les deux projets sélectionnés sont **Java Swing** (version 6 de *Sun Microsystems*) et **JFreeChart** (version 1.0.13) car ils respectent les sept contraintes.

Swing

Les classes de Swing (partie de *Java*™ *Foundation Classes* (JFC)) implémentent un ensemble de composants pour la construction d'interfaces utilisateurs graphiques (GUI) et ajoutent des fonctionnalités graphiques et interactives aux applications Java. Les composants de Swing sont entièrement intégrés dans le langage de programmation Java. Les "*Look and Feel*" *pluggables* permettent la création de GUIs semblables au travers de différentes plate-formes et rendent possible la prise en compte des "*Look and Feel*" des systèmes d'exploitations courants (tels que Microsoft Windows, Solaris™ ou GNU/Linux) [Ora]. L'entièreté du code de Swing est disponible (**C7**).

Swing est l'API la plus utilisée actuellement pour les interfaces graphiques des logiciels Java (hors applications Web) (**C1**) et chaque composant graphique qu'il implémente peut être isolé en un nombre de classes réduit (**C5**), toutes écrites en Java (**C2**,**C3**). Cette librairie n'a pas été utilisée dans une quelconque étude parmi les sujets (**C6**). Comme mentionné en Section 2.5, Swing utilise le patron architectural *Model-Delegate* (**C4**).

JFreeChart

JFreeChart [1] est une librairie Java qui simplifie l'affichage de graphiques dans les applications des développeurs. Les fonctions étendues de JFreeChart incluent [JFr09] :

- une API consistante et documentée qui supporte un grand nombre de types de graphiques ;
- une conception simple à étendre qui cible les applications côté-serveurs et les applications côté-clientes ;
- le support de plusieurs types de sorties, incluant des composants Swing, des fichiers images (PNG et JPEG) et des formats de fichiers vectoriels (PDF, EPS et SVG) ;

JFreeChart est à code ouvert (**C7**) et libre. Il est distribué sous les termes de la licence *Lesser General Public Licence* (LGPL), ce qui permet son utilisation dans des applications propriétaires.

JFreeChart est très utilisé par les développeurs Java, il en existe même des *plugins* tels que *Struts2-JFreeChart* (**C1**). Son composant graphique principal est facilement isolable (**C5**) et est intégralement écrit en Java (**C2**,**C3**). Comme mentionné en Section 2.5, JFreeChart utilise le patron architectural *Model-Delegate* (**C4**). Une seule étude utilisant cette librairie a déjà été soumise aux sujets de notre expérience. De plus, il s'agissait d'une étude ne visant pas l'analyse des patrons architecturaux mais l'analyse des anti-patrons [AKGA11] et opérant directement sur le code et non sur des diagrammes.

10.1.3 Réduction

Concernant Java Swing, le composant graphique représentant les tableaux de données illustre le principe des patrons architecturaux. Ce composant est utilisé via la classe `javax.swing.JTable`. La réduction d'un tel composant peut se faire en 28 classes. Outre la classe principale, toutes les interfaces des *listeners*, les classes d'éditions du tableau, les classes contenant les données et quelques

1. Cette description de JFreeChart provenant du site officiel, nous avons tenté d'en supprimer l'aspect "marketing". La traduction de la description n'est donc pas intégrale.

autres composants similaires au tableau ont été conservés. De plus, le concept de tableau à deux dimensions est une notion universelle pour n'importe quel utilisateur. Il s'agit là d'un avantage indéniable pour nos expériences.

La réduction du composant graphique principal de JFreeChart (représenté par `org.jfree.chart.JFreeChart`) peut être faite en 23 classes. Cet ensemble de classes comprend les *listeners*, les classes qui contiennent les données, les classes d'affichage et trois types de graphiques.

10.2 Patrons architecturaux

Afin de ne pas nécessiter un trop grand nombre de sujets, trois patrons couramment mentionnés dans la littérature ont été sélectionnés. Il s'agit des patrons **Modèle-Vue-Contrôleur** (MVC), **Modèle-Vue-Présentateur** (MVP) et ***Model-Delegate*** (MD) tels que définis en Section 2.5.

10.3 Représentation visuelle

Les parties de logiciels ou de *frameworks* qui sont soumises aux tâches de maintenance de l'étude doivent en premier lieu être modélisées avec un langage graphique. Face à ce besoin, deux questions se posent : "Que faut-il modéliser ?" et "Comment le modéliser ?".

10.3.1 Objet de la modélisation

Il existe deux aspects principaux à modéliser : l'aspect statique et l'aspect dynamique. Alors que le premier peut être facilement modélisé à l'aide de classes (dans le paradigme orienté objet), l'aspect dynamique est plus difficile à représenter. En effet, modéliser le comportement d'un système peut être fait de différentes façons : par les scénarios, par des règles de comportement plus générales, etc... Alors que les deux aspects sont nécessaires afin de caractériser l'intégralité d'un système, il nous est nécessaire de faire un choix. Il est plus commun, notamment dans le domaine des patrons en génie logiciel, de modéliser en priorité l'aspect statique.

10.3.2 Langage

Le choix du langage de modélisation graphique utilisé pour représenter les parties de programmes à étudier s'est imposé de lui-même. Le langage choisi est UML et plus particulièrement les diagrammes de classes. En effet, des diagrammes de classe sont tout à fait capable de représenter l'aspect statique d'un système conçu dans le paradigme orienté objet. De plus, son utilisation fortement répandue suppose qu'un plus grand nombre de sujets seront susceptibles de comprendre ce formalisme graphique.

10.4 Sujets

Les sujets ayant passé l'expérience sont majoritairement issus du laboratoire *Ptidej*[2] et du *Soccer Lab*[3] et plus généralement du département de Génie Logiciel de l'École Polytechnique de Montréal[4]. Trois étudiants des *FUNDP* ont également participé à l'expérience ainsi qu'une étudiante de l'université de Montréal. Une particularité quant aux sujets de notre étude est la répartition hommes/femmes qui est particulièrement homogène. Or, il s'agit là d'une situation relativement rare

2. `http://www.ptidej.net`
3. `http://web.soccerlab.polymtl.ca/`
4. `http://www.polymtl.ca`

dans le domaine de l'informatique.

Les sujets ont été recrutés par courrier électronique avec l'aide de Yann-Gaël Guéhéneuc, responsable du département. Un *doodle* [5] a été mis en place afin de permettre aux différentes personnes de choisir une plage horaire qui leur convienne le mieux.

10.4.1 Recherche

Dans le but d'obtenir un maximum de sujets pour participer à notre expérience, nous avons effectué des demandes personnalisées auprès de tous les membres du laboratoire *PtiDej* et *SoccerLab* avec l'aide de Yann-Gaël Guéhéneuc. Lui-même a effectué quelques demandes auprès de ses anciens élèves et collègues. En outre, nous avons plusieurs fois été sonder les laboratoires du département avec l'objectif de "recruter" des chercheurs ou des employés. Des demandes ont également été envoyées au Département d'Informatique et de Recherche Opérationnelle (DIRO) de l'Université de Montréal par l'intermédiaire de l'un de nos condisciples y effectuant un stage de recherche.

Les critères préliminaires (cf. Section 10.4.2) que doivent respecter les sujets ont été vérifiés oralement ou par courrier électronique afin de s'assurer que chaque personne intéressée pour passer notre expérience ait le minimum de capacités requises.

Étant donné les critères préliminaires, le temps nécessaire pour réaliser l'expérience avec un sujet et le temps qui nous était imparti pour faire passer un certain nombre de sujets, nous n'avons pas jugé nécessaire de faire d'appel public de type affiches ou annonces. Effectivement, les membres du laboratoire *Ptidej*, bien habitués à la recherche de sujets, nous ont affirmé que seules les demandes individuelles permettaient de recruter des sujets. En outre, beaucoup de personnes ont refusé de passer notre expérience par manque de temps ou d'expertise. La pression de notre maître de stage sur les membres du laboratoire, ainsi que le fait que nous ayons précédemment passé les études de certains membres, ont été nos meilleurs moyens de pression pour motiver ces personnes à passer notre expérience.

10.4.2 Critères préliminaires

Les critères préliminaires sont évalués oralement auprès du sujet potentiel. Il ne s'agit pas de questions formelles mais simplement de savoir si une personne lambda peut jouer le rôle de sujet pour notre expérience, ou non. La mortalité statistique ne tient pas compte des personnes ne respectant pas ces critères.

- Un sujet ne doit pas avoir de problèmes oculaires graves empêchant le fonctionnement de l'oculomètre.
- Un sujet doit être familier avec le paradigme orienté objet.
- Un sujet doit avoir une bonne connaissance d'UML [6].

10.4.3 Critères formels

Afin d'évaluer le niveau de connaissance des différents sujets ayant pris part à notre expérience, nous avons mis en place un questionnaire que chaque sujet a du remplir à la fin de la séance d'oculométrie. Ce questionnaire nous permet de recueillir les informations quant à leurs compétences en

5. `http://doodle.com/`
6. Une des chercheuses en informatique du département de GIGL nous ayant demandé si UML avait un rapport avec l'aéronautique (ULM), nous nous sommes préalablement renseignés sur les connaissances en UML des candidats.

UML et en patrons architecturaux. Cet aspect de l'expérience est expliqué plus en détail en Section 10.8.

Dans le but d'obtenir un échantillon uniforme de sujets en terme de compétences en patrons architecturaux et en UML, nous éliminons tous les sujets ne satisfaisant pas les critères minimum. Afin de réduire l'ensemble de sujets nous avons posé la contrainte suivante : *Les scores minimum d'un sujet en connaissances UML et en patrons architecturaux doivent tous deux être supérieurs ou égaux à 50% sous peine de rejet de l'ensemble des sujets.* Cette contrainte nous permet de nous assurer que tous les sujets ont un même niveau d'expérience et nous permet d'éliminer cette variable des observations effectuées.

Plusieurs expériences semblables effectuées précédemment fournissent un tutoriel sur UML avant l'expérience afin de considérer tous les sujets équivalents en terme de compétences dans ce domaine. Notre approche est différente car, à notre avis, un rapide tutoriel avant une expérience ne permet pas d'apporter un niveau de compétence suffisant au sujet pour pouvoir le considérer comme équivalent à ceux qui sont coutumiers de ce langage de modélisation.

Le questionnaire décrit en Section 10.8 devait obligatoirement être complété par tous les sujets ayant participé à l'expérience. Les données ont ensuite été utilisées afin d'éliminer certains sujets ayant une connaissance trop faible (en UML et en patrons) mais également afin d'attribuer un score à chaque sujet. Ce score nous a permis par la suite d'attribuer un niveau de connaissance à chaque sujet.

Grâce au questionnaire post-expérimental décrit en Section 10.8, les variables suivantes sont connues pour chaque sujet.

Niveau de connaissance UML : le niveau de connaissance en UML du sujet (dans l'intervalle $[0..2]$).

Niveau de connaissance en patrons architecturaux : le niveau de connaissance en patrons architecturaux (dans l'intervalle $[0..1]$). Il s'agit en réalité d'un questionnaire relatif à MVC.

Sexe : le sexe du sujet (`M | F`) (*Male – Female*).

Expérience en maintenance : si le sujet a déjà effectué des tâches de maintenance sur des diagrammes de plus de 20 classes (`1`) ou non (`0`).

Expert : si le sujet est considéré comme un expert (`1`) ou non (`0`). Cette valeur dépend du niveau de connaissance UML, en patrons architecturaux et selon son expérience de maintenance.

Niveau d'étude : le niveau d'étude du sujet (`U – B – M – D`) (*Inconnu – Bac – Master – Doctorat*).

Ces variables sont retranscrites dans les fichiers d'*input* de TAUPE comme décrit dans le guide de l'utilisateur (cf. Annexe D).

10.4.4 Réduction de l'ensemble des sujets

Suite aux problèmes rencontrés lors de l'expérience comme expliqués en Section 11.3.2 et suite à l'évaluation des compétences des sujets (UML et patrons architecturaux), les données de plusieurs sujets ont dû être mises de côté. Quatre types de problèmes mènent à quatre groupes de sujets qui doivent être éliminés selon les analyses pratiquées. Initialement, **21 sujets** ont participé à l'expérience.

Groupe A Les problèmes techniques liés à l'oculomètre empêchent, malgré le caractère exact ou non des réponses, d'enregistrer correctement les fixations. Alors que des modifications légères des fixations peuvent corriger certains *offsets*, l'oculomètre a enregistré majoritairement des *offsets*

chaotiques pour *un* sujet particulier. Ce sujet ne peut donc pas être pris en compte pour ses données oculométriques.

Groupe B Les logiciels d'enregistrement liés à l'oculomètre ont bogué de telle façon qu'un enregistrement d'un sujet n'a pas pu être conservé.

Groupe C Le questionnaire préliminaire nous a permis d'éliminer *trois* sujets ayant des connaissances en UML beaucoup trop faibles. Cette observation concernant le niveau de connaissance de ces sujets a été confirmée par l'analyse des réponses aux questions posées durant l'expérimentation. L'approche de notre expérience ne considère effectivement qu'un ensemble uniforme de sujets en terme de connaissance en UML. L'élimination de ces sujets est relatif au processus expérimental de *blocage* décrit en Section 6.3.3. Ce choix nous permet d'éliminer l'effet provoqué par des différences de compétences au sein des sujets.

Groupe D *Deux* sujets, pour des raisons toutes autres que celles mentionnées ci-dessus, n'ont pas pu participer à l'expérience.

Les données oculométriques des trois sujets éliminés pour cause de problèmes techniques ont néanmoins été utilisées pour les analyses de performance en terme d'exactitude des réponses (Section 12) et pour l'analyse pour l'analyse subjective de la charge mentale (Section 14). Les groupes concernés sont les groupes **B**, **C** et **D**. Par conséquent, 15 sujets sont pris en compte pour ces analyses et la mortalité est donc de 28,6% pour ces deux analyses.

Au final, les données oculométriques de 14 sujets ont été utilisées pour l'analyse des mesures physiologiques (Section 13) et les données pour la performance des sujets en terme de temps (Section 12). Les quatres groupes mentionnés ci-dessus sont concernés par la réduction de l'ensemble des sujets pour ces analyses. En conclusion, la mortalité de ces analyses est de 33 %.

10.5 Objets d'étude

Les tâches de maintenance effectuées par les sujets sur des diagrammes UML des projets JFreeChart et Swing (voir Section 10.1). Chacun de ces projets est modifié pour exister en trois exemplaires qui correspondent aux trois patrons architecturaux décrits en Section 10.2.

10.5.1 Obtention

La contrainte **C7** présentée en Section 10.1.1 impose que le code du projet soit ouvert et donc disponible. Alors que le code de Swing[7] a été récupéré au travers du package Debian `sun-java6-sources` disponible à l'adresse `http://packages.debian.org/sid/sun-java6-source`, le code source de l'API JFreeChart est disponible à l'adresse `http://sourceforge.net/projects/jfreechart/files/`).

Afin d'obtenir des diagrammes de classes UML, les codes sources ont été traités par **rétro-ingénierie** à l'aide de l'outil *Visual Paradigm For UML*[8]. Le choix de la rétro-ingénierie diminue la probabilité d'un éventuel biais introduit par l'expérimentateur.

7. version Java 6 de *Sun Microsystems*
8. `http://www.visual-paradigm.com/`

10.5.2 Adaptation des diagrammes

Plusieurs contraintes nous ont forcé à adapter les diagrammes générés par rétro-ingénierie. La première modification majeure était le **réarrangement des éléments** (classes et relations) des diagrammes car ceux-ci n'étaient pas disposés correctement. C'est-à-dire que certaines classes se chevauchaient, le diagramme n'était pas planaire. Ces problèmes d'affichage sont récurrents avec la technologie de rétro-ingénierie.

Les contraintes matérielles nous ont forcé à dimensionner les images afin qu'elles soient les plus appropriées à la **taille et à la résolution de l'écran disponible**. Le moniteur était un écran TFT *16/9* d'une résolution de 1920×1080 pixels avec une taille de 24 pouces. Ce moniteur relativement récent nous a donc permis de facilement disposer nos éléments et d'obtenir une bonne qualité d'image. Ce facteur est important car la taille élevée de chaque composant réduit le manque de précision de l'oculomètre. De plus, une **marge** a été ajoutée à chaque image afin que même en cas d'*offset*, l'oculomètre soit toujours en mesure de récupérer les données du regard (l'oculomètre supprimant les fixations enregistrées en dehors de l'image).

Tous les **accesseurs** tels que les *getters* et les *setters* ont été supprimés des diagrammes. Il en va de même pour les **constructeurs**. Seules les associations portant le nom *get state* et/ou *set state* permettent respectivement de donner accès aux *getters* et aux *setters* d'une classe à une autre.

De plus, le **"sens" des relations** n'est pas indiqué par les indicateurs de navigabilité UML mais par des flèches dans le nom de ces relations.

Nous considérions que le style de police en italique pour les classes et les méthodes abstraites n'étaient que peu visibles pour le sujet. Dans le but de rendre cette notion plus évidente, un **stéréotype** `Abstract` a été ajouté aux classes correspondantes. Le même principe a été appliqué aux interfaces par le stéréotype `Interface`.

Étant donné que les tâches à effectuer n'impliquaient jamais cette notion et par soucis de lisibilité des diagrammes, toutes les **cardinalités** ont été **masquées**. Il en va de même pour les rôles de relation. Néanmoins, tous les noms de relation ont été modifiés pour être les plus explicites possibles.
L'échelle de chaque diagramme est adaptée pour que tous les diagrammes aient une police d'écriture de taille équivalente.

10.5.3 Injection des patrons architecturaux

Les deux projets initiaux (Swing et JFreeChart) disposent d'un patron architectural *Model-Delegate* (MD). Les deux autres patrons à utiliser étant Modèle-Vue-Présentateur (MVP) et Modèle-Vue-Contrôleur (MVC) tels que définis en Section 2.5 doivent remplacer le patron architectural initial afin de former six diagrammes différents sur base des deux projets et des trois patrons architecturaux. Il est donc nécessaire de déterminer deux méthodes : une qui permette de passer d'un patron MD à MVC et de MD à MVP.

De MD vers MVC

En l'occurrence, plusieurs étapes sont nécessaires au passage d'un patron *Model-Delegate* à un patron Modèle-Vue-Contrôleur.

1. Détecter les classes V_i qui contiennent la définition d'une interface utilisateur et de la capture et de la gestion d'un ou plusieurs évènements.
2. Créer une classe `Controller`.
3. Supprimer toutes les méthodes relatives à la gestion d'évènements des classes V_i et les ajouter à la nouvelle classe `Controller`. Il est implicitement considéré que les noms des méthodes n'entrent pas en conflit au regard du paradigme orienté-objet. De la même façon, si les classes V_i implémentent des interfaces relatives aux *listeners*, la nouvelles classe `Controller` doit implémenter ces interfaces à la place des V_i.
4. Créer une association, de la classe `Controller` vers toutes les classes V_i et vers les classes du modèle, nommée `sets/gets state`.
5. Remplacer les associations `sets state` des classes V_i vers les classes du modèle.
6. Si elles n'existent pas, créer une association `notifies` des classes du modèle vers les classes V_i.
7. Ajouter des méthodes `link` à la classe `Controller` pour lui permettre d'enregistrer les vues comme observateurs du modèle.
8. Créer une association, des classes V_i vers la classe `Controller`, nommée `fire events`.

Il est évident que certaines modifications sont spécifiques au problème d'où la raison, en l'occurrence, d'effectuer ces étapes manuellement. Cela implique que les étapes mentionnées ci-dessus sont générales au problème de conversion d'un patron vers un autre sans pour autant être exhaustives.

De MD vers MVP

D'une manière générale, les étapes sont proches de celles décrites dans la section précédente. Néanmoins, la classe `Controller` est une classe `Presenter`. De plus, toutes les relations des classes V_i vers le modèle nommé `get state` transitent par cette nouvelle classe. En outre, si le `Presenter` gère effectivement les évènements, il n'implémente pas forcément les interfaces de *listener*.

10.5.4 Résultat

Après réarrangement des éléments dans le but de similarité graphique entre les trois diagrammes d'un même projet, certains éléments parasites ont été ajoutés. Par exemple, les classes `JButton` et `JTextField` ont été insérées dans les diagrammes relatifs au projet Swing. Il en va de même pour le projet JFreeChart où différents types de graphiques ont été intégrés au diagramme. En outre, les arguments de la méthode `draw()` présente dans les diagrammes de JFreeChart apparaît sans aucun paramètre par soucis de lisibilité et sans aucune perte d'information significative pour les tâches de maintenance.

Par exemple, la Figure E.6 illustre le patron MVP injecté dans le projet Swing et la Figure E.2 présente le patron MVC injecté dans le projet JFreeChart.

10.5.5 Similarité visuelle

Afin de savoir si deux diagrammes traitant d'un même projet mais utilisant deux patrons architecturaux différents sont visuellement proches, nous avons conçu une technique simple que nous

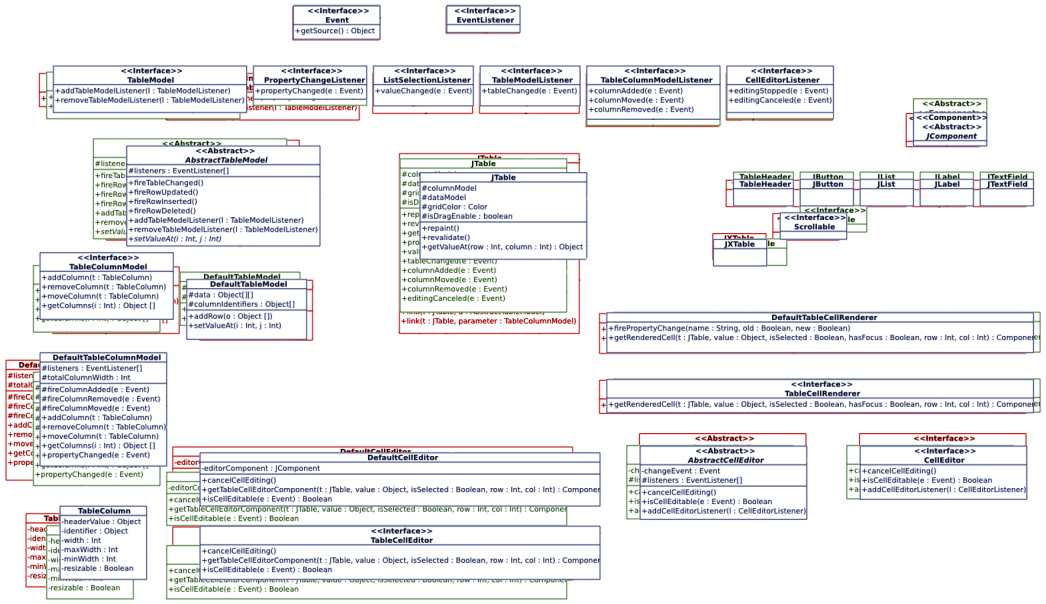


FIGURE 10.1 – Superposition des diagrammes MVC, MVP et MD du projet Swing/JTable

nommons “technique de la similarité visuelle”. La Figure 10.1 illustre la superposition des trois diagrammes de classes du projet Swing afin de donner une justification informelle de la similarité visuelle de ces diagrammes. Seules les classes communes aux trois diagrammes sont affichées. La Figure 10.1 est une première argumentation satisfaisante de la similarité visuelle des trois diagrammes. Cette sous-section a pour objectif de présenter une technique plus formelle conçue à cet effet.

Technique mise en place

Le principe général de cette technique est de comparer, pour un même projet mais pour des patrons architecturaux différents, les différences de disposition des classes en considérant la différence euclidienne deux par deux. Les étapes de cette technique, **pour un même projet**, sont les suivantes :

1. Considérer les classes communes entre les différents diagrammes du projet.
2. Calculer, pour chaque diagramme, les coordonnées (x, y) du centre de chacune des classes. Supposons donc un type `Classe` qui contient quatre attributs :
   - `nom` est le nom de la classe (ce nom est commun entre les différents diagrammes du projet)
   - `patron` est le patron architectural qui correspond au diagramme
   - `x` est la coordonnée en abscisse du centre de la classe
   - `y` est la coordonnée en ordonnée du centre de la classe
3. Éliminer, pour chaque diagramme, la longueur des marges des coordonnées du centre de chaque classe. C’est-à-dire qu’il est nécessaire de soustraire la largeur de la marge de gauche à l’abscisse des centres et soustraire la hauteur de la marge du haut à l’ordonnée des centres. Cette étape est obligatoire car les marges de chaque diagramme ne sont pas toujours identiques.
4. Calculer, pour tous les diagrammes, les distances euclidiennes de toutes les classes deux-à-deux.

Avec ces données, il est possible de comparer tous les diagrammes du même projet en terme de distances entre les classes. Par exemple, le calcul de l’écart-type des distances entre deux classes permet de savoir si la distance entre deux classes est constante, peu importe le patron architectural

Algorithme 2 : Algorithme de différenciation visuelle des diagrammes d'un même projet pour différents patrons architecturaux

Données :
P l'ensemble des patrons architecturaux
C l'ensemble des classes communes entre les différents diagrammes ($= \bigcap_{p \in P} classes(p)$)
Entrées :
`data : Classe[1..#P][1..#C]` les classes pour chaque type de patron pour chaque type
Résultat : L'écart-type des distances entre les classes deux-à-deux

```
resultInPix ← new integer[1..#P][1..#C][1..#C];
for i ← 1 to #C do
    for j ← 1 to #C do
        for k ← 1 to #P do
            resultInPix[k][i][j] ← √(|data[i].x − data[i].y|² + |data[j].x − data[j].y|²)
        end
    end
end
std ← new integer[1..#C][1..#C];
for i ← 1 to #C do
    for j ← 1 to #C do
        sum_std ← 0;
        sum ← 0;
        for k ← 1 to #P do
            sum_std ← sum_std + resultInPix[k][i][j] ∗ resultInPix[k][i][j];
            sum ← sum + resultInPix[k][i][j];
        end
        sum_std ← sum_std / #C;
        sum ← sum / #C;
        std[i][j] ← √(sum_std − sum ∗ sum);
    end
end
return std;
```

considéré. Cette technique est justifiée car les utilisateurs parcourent les diagrammes en naviguant de classe en classe. **Moins l'écart-type de la distance entre deux classes est élevé, au mieux les différents diagrammes sont visuellement proches**[9].

L'Algorithme 2 permet d'obtenir ces écart-types. La fonction $classes$ renvoie les classes relatives à un patron architectural donné.

Limites

Une menace évidente à cette technique est qu'un diagramme serait considéré équivalent face à sa version symétrique. Nous pourrions alors envisager le cas de comparer la position absolue de quelques éléménts de référence. Pour traiter les diagrammes soumis à notre expérience, ce mécanisme ne sera pas nécessaire.

En outre, cette approche ne tient compte que des classes communes aux trois diagrammes de chaque projet. C'est-à-dire que, par exemple, les classes `TableController` et `TablePresenter` ne sont

9. La technique qui consiste à vérifier entre elles les coordonnées du centre d'une classe dans les différents patrons architecturaux n'est pas représentative. En effet, un simple décalage de toutes les classes vers la droite, par exemple, biaiserait le résultat final. Il est donc nécessaire de comparer les classes les unes avec les autres.

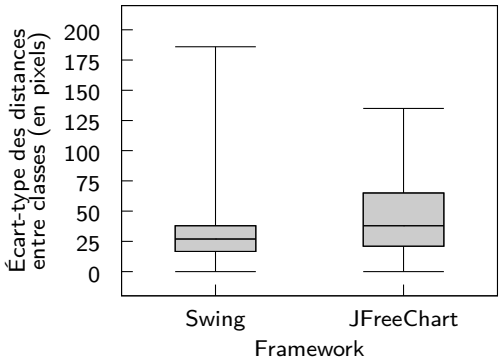


FIGURE 10.2 – Boîte à moustaches représentant les écart-types de la similarité visuelle des diagrammes de JFreeChart et Swing

pas prises en compte par la technique appliquée aux diagrammes de Swing. Néanmoins, ces classes influencent indirectement la position des autres, position qui est analysée.

Application de la technique

Le Tableau A.8 représente l'écart-type de la distance des classes deux-à-deux au travers des différents patrons architecturaux des diagrammes de JFreeChart . Le nom des classes, par soucis de lisibilité, est remplacé par des lettres. Dans chaque diagramme, la distance d'une classe avec elle-même est évidemment nulle. De part notre procédé, l'écart-type des distances d'une classe avec elle-même est donc nul également. Visuellement, il est évident que la classe `PopUpMenu` du diagramme supportant le patron *Model-Delegate* n'est pas disposé de la même façon que dans les deux autres versions de JFreeChart. Dans le Tableau A.8, la classe `PopUpMenu` est représentée par la lettre M. L'écart-type de la distance de M avec les autres classes est nettement supérieur à toutes les autres classes (la moyenne du tableau étant de 45, la moyenne des écart-types de M est de 89,8).

La moyenne des écart-types des distances est égale à 45 pixels, c'est-à-dire 2,35% de la largeur du diagramme et à 4,16% de sa hauteur. L'écart-type maximal est de 135 pixels, c'est-à-dire 7,03% de la hauteur du diagramme et 12,5% de sa hauteur. Au vu de ces comparaisons, nous pouvons affirmer que la disposition des éléments des diagrammes pour un même projet est fortement uniforme.

À l'instar de JFreeChart, les diagrammes de Swing/JTable sont fortement semblables. Effectivement, leur écart-type maximal est de 186 pixels (9,7% de la largeur et 17,2% de la hauteur) et la moyenne de tous les écart-types des distances est de 34,9 pixels (1,8% de la largeur du diagramme et 3,23% de sa hauteur). Seules les classes **C** et **D** (qui représentent respectivement `PropertyChangeListener` et `TableModel`) ont des écart-types de distance supérieurs aux autres classes car elles sont inversées dans le diagramme relatif au patron *Model-Delegate*. La boîte à moustaches illustrée en Figure 10.2 résume des écart-types pour les deux *frameworks*. Ce graphique met en exergue le fait que les écart-types, au sein d'une même *framework*, sont répartis sur des données de valeurs insignifiantes au vu de la taille totale des images.

10.5.6 Similarité en complexité

Yi et al. énoncent plusieurs ensembles de métriques afin d'évaluer la complexité et la maintenabilité d'un diagramme de classes UML . Par exemple, ils énumèrent les métriques proposées par Marchesi, Genero, In, Rufai, Zhou et Kang [YWG04].

Cette section a pour objectif d'évaluer les différents diagrammes d'un même projet selon les métriques de Genero telles que définies dans [GOPR01].

Métriques de Genero

Genero et al. [GOPR01] classent ses métriques en deux catégories : les métriques *open-ended*, dont les valeurs ne sont pas bornées dans un intervalle, et les métriques *close-ended* dont les valeurs sont bornées dans l'intervalle $[0, 1]$.

Les métriques **open-ended** sont [GOPR01, p. 4] :

Nombre total de classes (NC) Intuitivement, le nombre total de classes est un indice simple qui, lorsqu'il croît, augmente la complexité d'un diagramme.

Nombre total d'attributs (NA) Le nombre d'attributs disponibles affecte la taille des classes, il devrait être faible [GPC00, p. 7].

Nombre total de méthodes (NM) Le nombre de méthodes disponibles affecte la taille des classes, il devrait être faible [GPC00, p. 6].

Nombre total d'associations ($NAssoc$) La complexité d'une classe dépend du nombre d'associations qu'elle a avec d'autres. La complexité d'une classe dépend de ce nombre [GPC00, p. 20].

Nombre total d'agrégations ($NAgg$) Un nombre plus élevé de relations d'agrégation constitue une complexité de conception plus élevée. Par conséquent, cela peut nécessiter un cout plus élevé en implémentation et en maintenance [GPC00, p. 23].

Nombre total de dépendances ($NDep$) Plus le nombre de classes qui dépendent d'une autre est élevé, plus les dépendances inter-classes sont nombreuses et par conséquent, plus la complexité de la conception est élevée [GPC00, p. 24].

Nombre total de généralisations ($NGen$) Il s'agit du nombre de relations "père-fils" entre les classes.

Nombre total de hiérarchies de généralisation ($NGenH$) Plus grand est le nombre de hiérarchies d'héritage, plus la complexité d'un diagramme est élevée.

DIT Maximum Cette métrique correspond au maximum du DIT (*Depth Inheritance Tree*) de chaque classe du diagramme. La valeur du DIT obtenue pour chaque classe est la longueur du chemin de la classe jusqu'à la racine de sa hiérarchie. Basili conclut, après une validation empirique, que plus le DIT est élevé, plus grande est la probabilité de détecter des fautes [GPC00, p. 4].

Les métriques **close-ended** sont [GOPR01, p. 4] :

Nombre d'associations VS nombre de classes ($NAssocVC$) Plus un diagramme a d'associations par classe, plus il est complexe et plus il est difficile à maintenir.

$$NAssocV = \frac{NAssoc}{NC}$$

	JFreeChart			Swing		
	MD	MVC	MVP	MD	MVC	MVP
NC	21	23	23	28	29	29
NA	25	23	23	17	17	17
NM	34	33	35	60	60	62
NAssoc	12	12	14	9	11	13
NAgg	0	0	0	1	1	1
NDep	0	0	0	0	0	0
NGen	19	22	22	26	26	26
NGenH	2	3	3	8	8	7
MaxDIT	3	2	2	3	2	3
NAssocVC	0.57143	0.52174	0.60870	0.32143	0.37931	0.44828
NDepVC	0	0	0	0	0	0
NAggVC	0	0	0	0.03571	0.03448	0.03448
NGenVC	0.90476	0.95652	0.95652	0.92857	0.89655	0.89655

TABLE 10.1 – Comparaison des métriques de Genero et al. [GOPR01] sur les six diagrammes différents

Nombre de dépendances VS nombre de classes ($NDepVC$) L'analyse de cette métrique est identique à $NAssocVS$.

$$NDepVC = \frac{NDep}{NC}$$

Nombre d'agrégations VS nombre de classes ($NAggVC$) L'analyse de cette métrique est identique à $NAssocVS$.

$$NAggVC = \frac{Agg}{NC}$$

Nombre de généralisations VS nombre de classes ($NGenVC$) L'analyse de cette métrique est identique à $NAssocVS$.

$$NGenVC = \frac{NGen}{NC}$$

Application des métriques

Le Tableau 10.1 présente la valeur des métriques pour les six diagrammes. Il est évident que la plupart des métriques sont fortement équivalentes et indiquent par conséquent un niveau de maintenabilité équivalent en terme de complexité de diagramme [10].

10.6 Instrumentation

Afin de mesurer la charge de travail et ainsi évaluer quel patron architectural demande la charge de travail la plus faible, nous avons eu recours à différents instruments. Ceux-ci sont détaillés dans les sections suivantes.

10.6.1 Choix de l'oculomètre

Le fait que nous ayons à notre disposition un oculomètre performant, que les mesures physiologiques soient pertinentes pour l'évaluation de la charge mentale (tel que détaillé en Section 5.1) et que l'utilisation de l'oculométrie en génie logiciel soit un domaine encore peu exploité et soit la spécialité du laboratoire *PtiDej* ont motivé l'utilisation d'un oculomètre dans notre expérience.

10. Le nombre d'associations de MVC et principalement de MVP est plus élevé que pour MD. Cela peut s'expliquer par le fait qu'une décomposition de classes selon différents rôles provoque la nécessité d'assurer ces rôles au travers de nouvelles associations.

Afin de pouvoir mesurer les mouvements oculaires et tenter d'estimer la charge mentale des différents sujets passant notre expérience, l'oculomètre choisi est *FaceLAB* (de *Seeing Machine*) décrit en Section 8.1.2. Ce choix est motivé par le fait que ce système est plus récent, plus précis, moins intrusif et plus facile d'utilisation que l'Eye-link®II. Effectivement, le sujet peut notamment bouger la tête sans que l'étape de calibration ne doive être refaite. Les outils logiciels fournis avec le système sont également plus performants, proposent une analyse préliminaire des données et la création d'une expérience est grandement facilitée par le logiciel. En outre, il nous a été demandé d'utiliser *FaceLAB* étant donné le coup d'investissement du laboratoire dans cet appareil. Le fait que ce nouveau système ne nécessite qu'un seul ordinateur (fixe ou portable) est également un avantage notable pour le déroulement de l'expérience.

10.6.2 Support d'interrogation

Plusieurs tâches sont données au sujet quant à la maintenance de diagrammes apparaissant à l'écran. Contrairement à l'approche de Van Den Plas [Van09], les questions ne sont pas affichées à l'écran mais sont imprimées dans le questionnaire qui est fourni au sujet en début d'expérience. Le déroulement de l'expérience (tel que décrit en Section 11.3) permet au sujet de prendre connaissance de la question à son aise et ensuite d'y répondre sur son document. En cas de doute du sujet, l'oculomètre choisi lui offre la possibilité de porter son regard sur le document sans biaiser les données enregistrées par l'appareil. L'utilisation d'un support papier peut éventuellement poser problème s'il le place devant lui et donc potentiellement en face des caméras de l'oculomètre. La disposition du laboratoire présenté dans la section suivante permet de résoudre ce type de problèmes. Le questionnaire est décrit plus en détail en Section 10.7.2.

10.6.3 Environnement

La Figure 10.3 détaille l'agencement du laboratoire pour les expériences que nous avons menées. Cet agencement est capital pour le déroulement de l'expérience, il a d'ailleurs été intégralement choisi selon nos besoins. La zone de travail est découpée en deux zones principales :

- la zone notée (A) est la zone réservée aux sujets
- la zone (E) est celle de l'expérimentateur

Une chaise de dentiste (positionnée en (J)) introduite durant les expériences de Van Den Plas [Van09] est utilisée pour s'assurer que le sujet bouge peu durant l'expérience. En effet, même si l'oculomètre permet des mouvements de la part du sujet, son immobilité garantit de meilleurs résultats. Le PC portable utilisé pour enregistrer les données (H) possède un deuxième écran branché (B) afin d'afficher les diagrammes au sujet. L'écran présenté à l'utilisateur est un écran 24 pouces 16/9 d'une résolution de 1920×1080 pixels pour une meilleure lisibilité et un plus grand confort visuel. L'autre écran permet à l'expérimentateur de superviser les différentes métriques, la qualité du suivi du regard et lui permet de s'assurer que le sujet ne quitte pas le champ des caméras (C) ou ne mette ses mains, ou le support papier devant les caméras. Le sujet peut parcourir les questions en utilisant le clavier situé en (D). L'ordinateur (F) et son écran (G) sont utilisés pour enregistrer les vidéos prises par la caméra de scène située en (I) qui nous permet d'avoir une vidéo de l'ensemble de l'expérience. La Figure 10.4 est une photo d'un sujet passant l'expérience.

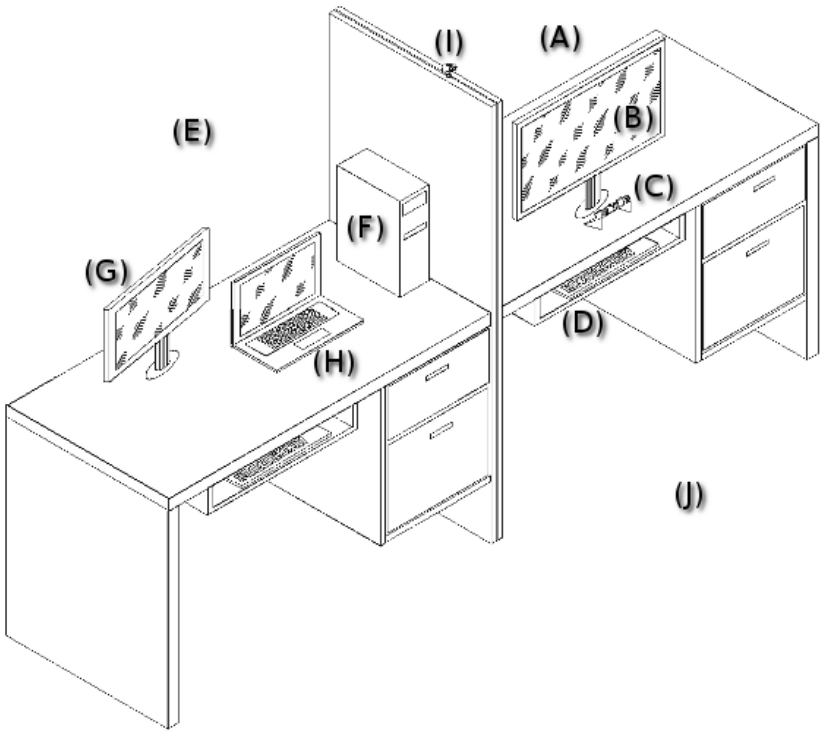


FIGURE 10.3 – Agencement du laboratoire pour notre expérience

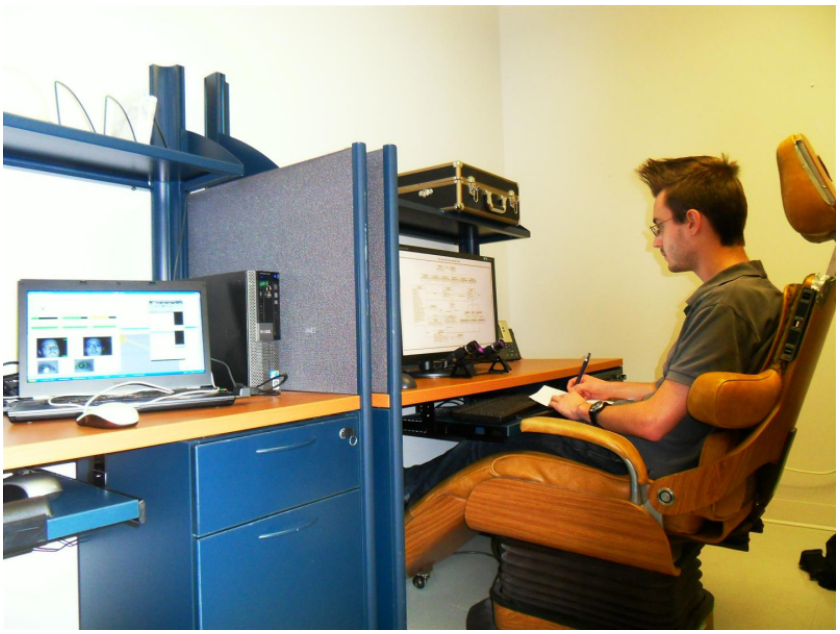


FIGURE 10.4 – Photo d'un sujet passant l'expérience

10.7 Tâches de maintenance

Sur base des diagrammes affichés au sujet, des questions doivent être posées dans le but qu'il réalise certaines tâches.

10.7.1 Portée des questions

L'objectif final des tâches à réaliser est évidemment d'effectuer une maintenance du diagramme associé à la question posée. Le terme "maintenabilité" est considéré dans notre expérience selon la définition du modèle de qualité ISO9126 décrit en Section 3.1. C'est-à-dire que la maintenance interne est composée de cinq critères de qualité que sont l'analysabilité, la changeabilité, la stabilité, la testabilité et la conformité.

La stabilité n'est pas considérée dans notre expérience car une réponse à une question ne sera considérée valide que si elle atteint au minimum le même niveau de stabilité de l'objet d'étude avant la tâche de maintenance. Le format des réponses demandées aux sujets ne permet pas d'étudier une quelconque conformité à un standard ou à une convention, ce facteur de qualité ne sera donc pas pris en compte. En outre, les réponses ne sont considérées valides que si la testabilité n'est pas affectée par les tâches du sujet.

Par conséquent, les deux critères qui mènent à deux types de questions sont l'analysabilité et la changeabilité. Nous nommerons respectivement les questions qui leur correspondent par "**question de compréhension**" et "**question de modification**".

10.7.2 Questionnaire

Le questionnaire donné aux sujets passant l'expérience est disponible en Annexe F. Celui-ci se compose d'une première partie permettant d'expliquer clairement au sujet la procédure, les termes et les concepts spécifiques à notre expérience.

Une des particularités du vocabulaire de notre questionnaire concerne le "chemin logique de l'information" parfois demandé dans certaines questions. Cette expression est utilisée pour parler de la "hiérarchie d'appel" des différentes classes. Cette notion est clairement expliquée au début du questionnaire et un exemple est également fourni. Nous étions à la disposition de tous les sujets et nous nous assurions que ceux-ci avaient bien compris toutes les notions expliquées en début du questionnaire avant de procéder à l'expérience proprement dite.

La suite du questionnaire comporte les différentes questions/tâches à réaliser sur les diagrammes. Chaque diagramme est introduit par un paragraphe qui permet de replacer le diagramme dans son contexte.

Notons que le questionnaire était disponible en anglais et en français afin de nous assurer que tous les sujets comprennent bien toutes les questions. Nous avons cependant choisi de privilégier la version anglophone (même pour les sujets francophones) afin de mettre tous les sujets sur le même pied d'égalité et également afin de limiter le biais lié à la traduction des termes contenus dans le diagramme. En effet, les diagrammes étant en anglais, il est plus simple pour un sujet de repérer directement un terme identique sur le diagramme que de devoir chercher son équivalent dans l'autre langue.

L'ordre des questions a été spécialement choisi afin de placer les questions de compréhension avant celles de modification. Cela permet au sujet d'explorer le diagramme, de comprendre le fonctionnement du logiciel pour seulement ensuite effectuer les tâches de modification. Ce choix nous permet

	Question 1			Question 2			Question 3			Question 4			Question 5			Question 6		
	D	C	P	D	C	P	D	C	P	D	C	P	D	C	P	D	C	P
Question	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Event	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EventListener	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ActionListener	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
ChartChangeListener	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
ChartProgressListener	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
ChartMouseListener	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
TitleChangeListener	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
DataSetChangeListener	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
ChartPanel	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0
JFreeChart	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
Plot	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	1	0
PopUpMenu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DataSet	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AbstractDataSet	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
DataSetGroup	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CategoryPlot	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0
PiePlot	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0
ThermometerPlot	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0
DefaultCategoryDataSet	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
DefaultPieDataSet	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
DefaultValueDataSet	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
View	■	0	0	■	0	0	■	0	0	■	0	0	■	0	0	■	0	0
ChartController	■	0	■	■	0	■	■	1	■	■	0	■	■	1	■	■	0	■
ChartPresenter	■	■	0	■	■	0	■	■	0	■	■	0	■	■	0	■	■	1

TABLE 10.2 – Attribution des zones d'intérêt de chaque tâche pour le diagramme de JFreeChart

de contrôler la courbe d'apprentissage et ainsi de réduire cette menace à la validité de notre expérience.

10.7.3 Zones d'intérêts

Afin de pouvoir évaluer l'influence des variantes de MVC par l'oculométrie, il a été nécessaire d'associer à chaque zone d'intérêt[11] une pertinence comme décrit en Section 4.2.3. Nous avons considéré une zone d'intérêt (classe) comme pertinente si cette classe doit être modifiée ou visualisée pour répondre à la question/tâche demandée.

Il est donc logique que les zones d'intérêt puissent être différentes en fonction de la variante de MVC qui lui est appliquée. De manière générale, la zone relative au numéro de la question sur chaque diagramme est considérée comme une zone ignorable. Les Tableaux 10.3 et 10.2 définissent la pertinence des zones d'intérêt pour chaque alternative de MVC pour chaque question respectivement pour les projets Swing et JFreeChart. Le patron MD y est représenté par la lettre **D**, le patron MVC par la lettre **V** et le patron MVP par la lettre **P**. Une cellule noire signifie qu'une zone ne correspond pas à un diagramme donné, le caractère **-** signifie que la zone est ignorable, le chiffre **1** que la zone est pertinente et **0** qu'elle ne l'est pas.

11. Dans le cadre de notre expérience toute classe est une zone d'intérêt.

	Question 1			Question 2			Question 3			Question 4			Question 5			Question 6		
	D	C	P	D	C	P	D	C	P	D	C	P	D	C	P	D	C	P
Question	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Event	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EventListener	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PropertyChangeListener	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TableModel	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ListSelectionListener	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TableModelListener	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TableColumnModelListener	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
CellEditorListener	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
AbstractTableModel	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
JTable	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
TableColumnModel	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
DefaultTableModel	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
JComponent	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TableHeader	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JButton	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JList	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JLabel	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
JTextField	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scrollable	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JXTable	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
DefaultTableCellRenderer	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	1	1	0
TableCellRenderer	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	1	1	0
DefaultTableColumnModel	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
TableColumn	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
DefaultCellEditor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TableCellEditor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AbstractCellEditor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CellEditor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TableController		0			1			0			1			0			0	
TablePresenter			0			1			0			1			1			1

TABLE 10.3 – Attribution des zones d'intérêt de chaque tâche pour le diagramme de Swing/JTable

10.8 Questionnaire post-expérimental

Afin de pouvoir évaluer le niveau de connaissance des différents sujets passant notre expérience, nous avons mis en place un questionnaire post-expérimental utilisant LimeSurvey [12]. Ce questionnaire est composé de 18 questions divisées en quatre parties et est disponible en Annexe G. Le questionnaire était disponible en anglais et en français et, comme toutes les autres données récoltées, celles-ci étaient totalement anonymes.

La première partie du questionnaire concerne les données d'identification du sujet. Il lui est demandé le numéro de sujet qui lui a été attribué, son sexe et son niveau d'étude. Ces données sont utilisées afin d'étudier l'influence de ces facteurs sur les résultats enregistrés (voir Sections 12, 13 et 14). La majorité des questions sont à choix multiples afin de faciliter la correction et d'aider les sujets qui pourraient ne pas se souvenir exactement de la réponse. Dans le but de s'assurer que le sujet ait bien répondu à toutes les questions, celles-ci ont été rendues obligatoires et nous avons ajouté un choix "pas de réponse" au cas où le sujet ne connaîtrait pas la réponse (pour éviter qu'il ne réponde au hasard).

La seconde partie évalue le niveau de connaissance du patron architectural MVC dans sa forme canonique, la troisième le niveau de connaissance de la syntaxe UML, la quatrième partie s'intéresse à l'expérience personnelle du sujet et la dernière partie permet au sujet de laisser des commentaires concernant l'expérience. Ceux-ci nous ont permis de récolter un retour des sujets et de nous aider à améliorer le processus expérimental dans la mesure du possible.

12. LimeSurvey (`http://www.limesurvey.org/`) est un outil de sondage en ligne.

CHAPITRE 11

Mise en place

La connaissance s'acquiert par l'expérience, tout le reste n'est que de l'information.

Albert Einstein

UNE fois les éléments constitutifs de l'expérience détaillés, l'expérience doit être organisée et configurée. Premièrement, les Sections 11.1, 11.2, 11.3 présentent respectivement la définition de l'expérience, sa planification et son exécution selon les termes de Wohlin et al. [WRH+99] tels que détaillés en Section 4.3. Ensuite, les aspects et les contraintes liés à l'éthique de l'expérience sont présentés en Section 11.4.

Sommaire

11.1 Définition

La définition de l'expérience peut être définie telle que :

Analyse des patrons architecturaux Modèle-Vue-Contrôleur (MVC), Modèle-Vue-Présentateur (MVP) et *Model-Delegate* (MD)

Dans le but d' évaluer l'impact de ces différentes alternatives

Vis-à-vis de leur maintenabilité interne du produit logiciel

Du point de vue des développeurs

Dans le contexte d' étudiants de Master et de Doctorat du Département de Génie Logiciel de l'École Polytechnique de Montréal.

Objet d'étude Les patrons architecturaux sont présentés tels qu'injectés dans des diagrammes de classe UML de parties de projets réels. Ces diagrammes sont décrits en Section 10.5.

But L'objectif de l'expérience est la comparaison de ces patrons architecturaux les uns avec les autres.

Focalisation de la qualité L'impact étudié est l'effet qu'ont les patrons architecturaux sur la maintenabilité interne des programmes comme définie en Section 3.2.

Perspective La maintenabilité interne est considérée du point de vue des développeurs (cf. glossaire pour une définition) et plus particulièrement des mainteneurs de logiciels.

Contexte Les sujets sont des étudiants en Génie Logiciel dans le cadre du Master ou d'un Doctorat à l'École Polytechnique de Montréal, de la Faculté d'Informatique de Namur ou du département d'Informatique et de Recherche Opérationnelle (DIRO) de l'Université de Montréal. L'environnement est un laboratoire de l'École Polytechnique de Montréal dédié au déroulement de l'expérience. La description du choix des sujets est détaillée en Section 10.4.

11.2 Planification

La plupart des éléments qui constituent la planification de l'expérience sont détaillés au Chapitre 10. Cette section est structurée selon la planification de l'expérience décrite par Wohlin et al. [WRH+99].

11.2.1 Formulation des hypothèses

Notre approche emploie trois types d'analyses qui sont : l'analyse de la performance, l'analyse des mesures physiologiques et l'analyse des mesures subjectives. Par soucis de lisibilité, les hypothèses sont présentées dans les chapitres dédiés à ces analyses, c'est-à-dire respectivement les Chapitres 12, 13 et 14.

11.2.2 Sélection des variables

Deux variables indépendantes sont utilisées pour la conception de l'expérience : le patron architectural et le *framework* considéré. Trois traitements sont utilisés pour le patron architectural : MVC, MVP et MD (cf. Section 10.2). Deux traitements sont considérés pour le *framework* étudié : Swing et JFreeChart (cf. Section 10.1). Le fait de distinguer les deux *frameworks* différents est nécessaire afin d'identifier réellement d'où provient réellement l'influence sur les variables indépendantes.

Pour chaque type d'analyse apportée dans la suite de ce document, les variables dépendantes sont spécifiques. Les chapitres relatifs à ces analyses les présentent en temps utile.

11.2.3 Sélection des sujets

La Section 10.4 décrit la recherche, la sélection et la classification des sujets de l'expérience.

11.2.4 Conception de l'expérience

Au regard des deux variables indépendantes choisies, la conception est de type "conception $2*2$" selon la Section 6.3.3 avec la particularité que nous avons ici deux facteurs, l'un avec trois traitements et le second avec deux traitements. Nous généralisons en premier lieu les besoins en équilibrage et en randomisation (comme décrits en Section 6.3.3) avant d'appliquer ces contraintes pour obtenir la configuration de l'expérience.

Généralisation de la randomisation et de l'équilibrage

Afin de maximiser l'entropie de la distribution des traitements aux différents sujets tout en s'assurant que tous les traitements ont été distribués un même nombre de fois (équilibrage), nous généralisons ce problème avant de l'appliquer au cas de notre expérience.

Soient

A l'ensemble des sujets

N le nombre d'*attributions* données à chaque sujet

P l'ensemble des projets (traitements)

E l'ensemble de patrons architecturaux considérés dans l'expérience (traitements)

$f : A \to P \times S \times E$ fonction qui retourne, pour un sujet donné, l'ensemble de ses *attributions* [1]

Une *attribution* est un couple $(projet, patron) \in P \times E$. Initialement, le nombre d'*attributions* disponibles pour un sujet est égal à $\#P * \#E$ (où # correspond à la cardinalité d'un ensemble). Par conséquent, le nombre de combinaison d'*attributions* pour un sujet est égal à $(\#P * \#E)^N$.

Néanmoins, l'objectif d'entropie mentionné ci-dessus pose plusieurs contraintes. En l'occurrence, deux contraintes doivent être respectées en supposant que $\#T > 1 \wedge \#P > 1$:

C1 Chacune des attributions d'un sujet doit concerner un projet différent.

$$\forall a \in A \quad \forall (p_1, s_1, e_1), (p_2, s_2, e_2) \in f(a) : (p_1, s_1, e_1) \neq (p_2, s_2, e_2) \Rightarrow p_1 \neq p_2$$

C2 Chacune des attributions d'un sujet doit concernent un patron architectural différent.

$$\forall a \in A \quad \forall (p_1, s_1, e_1), (p_2, s_2, e_2) \in f(a) : (p_1, s_1, e_1) \neq (p_2, s_2, e_2) \Rightarrow e_1 \neq e_2$$

Ces contraintes impliquent que pour une *attribution* a pour un sujet s, il existe $(\#P - 1) * (\#S - 1) * (\#E - 1)$ *attributions* possibles pour ce sujet.

Configuration de l'expérience

En spécifiant les éléments décrits dans la Section 11.2.4, il est possible d'établir une configuration pour l'attribution de couples $(projet, patron)$ aux sujets. L'ensemble des traitements P contient 2

1. Par attribution, nous entendons un couple de traitements (patron,projet) attribué à un sujet.

	JFreeChart			Swing		
Subject	MVC	MVP	MD	MVC	MVP	MD
1	■				■	
2		■				■
3			■	■		

TABLE 11.1 – Configuration des attributions des couples $(projet, patron)$

éléments : Swing et JFreeChart. L'ensemble des traitements E contient 3 éléments : MVC, MVP et MD. Pour éviter une durée d'expérience trop élevée, nous avons décidé de fixer le nombre d'*attributions* N à 2. En conclusion, **pour que chaque *attribution* soit assignée au moins une fois, le nombre de sujet minimum est de 3**. Le Tableau 11.1 présente la distribution des couples $(projet, patron)$ qu'il est nécessaire de respecter afin de réduire l'effet d'apprentissage des sujets (Section 15.2).

11.2.5 Évaluation de la validité

Les menaces à la validité sont présentées en Section 15.2.

11.3 Exécution

L'expérience se déroule comme présenté en Figure 11.1. Chaque sujet se voit attribuer une variante (MVC, MVP, MD) de chacun des deux projets (JFreeChart et Swing). Pour chaque diagramme qui lui est attribué, une série de tâches de compréhension et de modification lui est affectée. Avant de procéder à l'expérience proprement dite, la calibration est effectuée. Nous procédons ensuite à une simulation pour nous assurer que l'étape de calibration s'est bien déroulée. Chaque type de projet est introduit par une petite description du projet comme visible dans le questionnaire présenté en Annexe F. Une fois que le sujet a lu la description du projet, il peut prendre connaissance de la première question.

Lorsque le sujet a bien lu et a bien compris la question, il peut afficher le diagramme et y répondre. Une fois qu'il a trouvé la réponse, le sujet peut arrêter l'enregistrement et poser sa réponse sur papier. Lors de l'écrire de sa réponse, le diagramme est toujours affiché afin de l'aider à se remémorer certaines informations telles que les noms de méthodes. Quand toutes les questions ont été répondues, le deuxième système est présenté au sujet, la simulation est refaite pour s'assurer que l'appareil est toujours calibré et la réponse aux questions peut commencer. Le déroulement d'une séance complète est détaillé dans la section suivante.

11.3.1 Étapes de l'expérience

Une séance se déroule habituellement comme suit :

1. Le sujet se présente à l'heure prévue dans le local dédié aux expériences oculométriques.
2. Il est accueilli, une boisson lui est proposée et un questionnaire lui est remis. Il est invité à s'asseoir et à prendre connaissance des premières pages du questionnaire expliquant les spécificités des questions et du déroulement de l'expérience. Il lui est possible de poser toutes les questions qu'il désire.
3. Le sujet est invité à prendre place dans la chaise de dentiste et la calibration de l'oculomètre est effectuée.

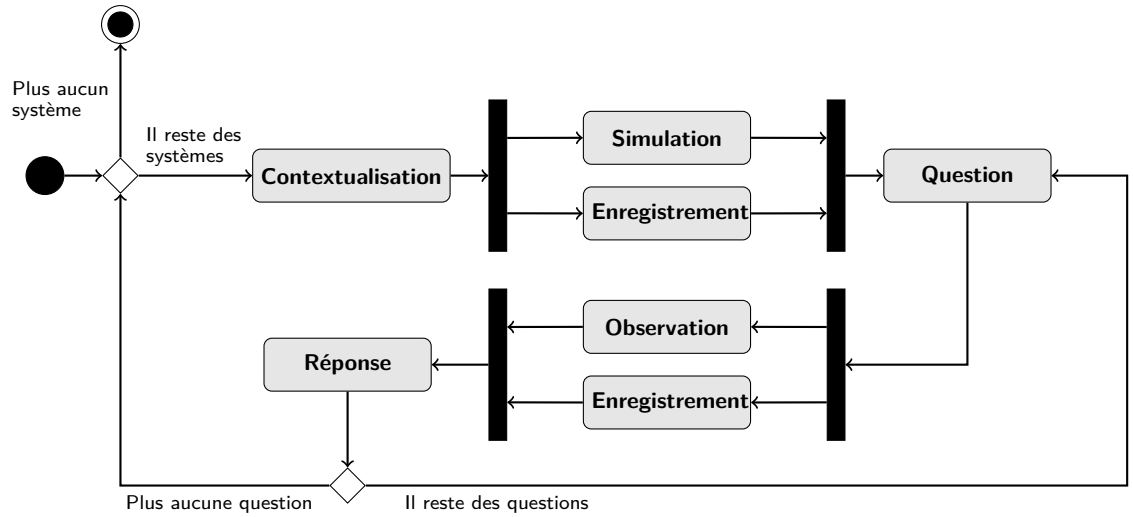


FIGURE 11.1 – Procédure expérimentale de notre expérience

4. Le sujet peut ensuite prendre connaissance de la mise en contexte du premier projet (JFreeChart).
5. Une fois prêt, le sujet peut lire la première question et s'assurer de bien l'avoir comprise. Nous étions présents tous les deux en permanence durant les séances et prêts à répondre à toutes les questions.
6. Une fois que le sujet a bien compris la question, celui -ci appuie sur la touche `ENTER` du clavier, le diagramme est affiché et l'enregistrement commence.
7. Dès que le sujet pense avoir trouvé la réponse, celui-ci appuie sur la touche `ENTER` du clavier et écrit sa réponse sur papier. Le diagramme est toujours présent à titre de support afin que le sujet n'aie pas a retenir la réponse par cœur. L'enregistrement continue durant cette période afin de s'assurer que le sujet ne cherche plus la réponse mais se contente d'utiliser le diagramme comme support à sa réponse.
8. Lorsque le sujet a fini de répondre, il appuie sur `ENTER` et peut commencer à lire la deuxième question.
9. Le processus continue de manière analogue jusqu'à ce que toutes les questions soient répondues.

Une fois que toutes les questions d'un type de diagramme étaient terminées, il était demandé au sujet d'évaluer sa charge mentale subjective à l'aide d'un questionnaire TLX disponible en Section F. Le sujet devait évaluer sa charge mentale à l'aide de différentes sous-échelles décrites en Annexe 5.3. Rappelons que les questionnaires étaient disponibles en français et en anglais et que les sujets avaient le choix de répondre dans la langue de leur choix. En effet, certains sujets étaient exclusivement anglophones.

11.3.2 Problèmes rencontrés durant l'expérience

Quelques soucis techniques ont été à déplorer durant les expérimentations. Le logiciel *GazeTracker* a malheureusement bogué à deux reprises et les données oculométriques de l'expérience associée ont été perdues. Les résultats d'un autre sujet ont également été perdus à cause d'une défaillance de l'enregistrement. De plus, certaines imprécisions ont été détectées sur les sujets portant des lunettes. En effet, la présence d'un reflet sur les lunettes empêche l'oculomètre de bien détecter le reflet de la

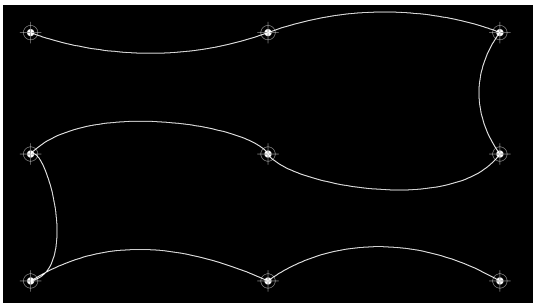


FIGURE 11.2 – Image utilisée afin d'évaluer l'*offset* de chaque sujet

cornée dans certains cas (voir Figure 4.2). Il était nécessaire de surveiller en permanence l'expérience étant donné le fait que beaucoup de sujets ont tendance a bouger hors du champ de la caméra ou de mettre le questionnaire devant la source infra-rouge.

Les *offsets* (voir Section 8.1.3) sont présents sur la presque totalité des enregistrements. Par conséquent, il est nécessaire de corriger certaines fixations. *GazeTracker* propose un outil permettant d'éditer manuellement les fixations pour les ajuster en fonction d'un modèle. Afin de clairement identifier les *offsets* et de pouvoir créer ce modèle, nous avons créé une image comportant neuf points clairement visibles (voir Figure 11.2). Avant chaque expérience, il est demandé au sujet de fixer chaque point durant quelques secondes. Les résultats de cette image nous permettent d'évaluer l'*offset* pour chaque section de l'image et, ainsi, de corriger facilement et précisément les fixations.

Notons que le déplacement manuel des fixations sur l'image est limité à un certain rayon par *GazeTracker* afin d'éviter de modifier trop fortement la position des fixations. Nous attirons l'attention sur le fait que si cette opération n'était pas restreinte, elle impliquerait un gros risque de biais introduit par la personne corrigeant les fixations. En effet, celle-ci pourrait avoir tendance à déplacer les fixations sur des zones d'intérêt alors que celles-ci étaient réellement en dehors.

Nous avons dû écarter un sujet qui portait un voile intégral et qui, par conséquent, rendait impossible le calibrage de l'oculomètre. Effectivement, FaceLAB nécessite la reconnaissance faciale du sujet afin de le détecter dans l'espace.

Ces problèmes impliquent la réduction de l'ensemble des sujets comme la Section 10.4.4 la décrit.

11.4 Aspects éthiques

La notion d'éthique est intervenue au moment où nous avons émis la possibilité de faire passer notre expérience à des étudiants effectuant leurs études à l'École Polytechnique. Un des avantages que nous mettions en avant était la possibilité d'évaluer directement le niveau d'un étudiant simplement en analysant les résultats scolaires de celui-ci. Néanmoins, cette démarche n'est pas autorisée par le comité d'éthique de l'Université et nous avons donc abandonné l'idée d'utiliser les résultats scolaires des étudiants pour notre expérience.

Quelques craintes ont également été émises de la part des membres du laboratoire concernant l'utilisation des données collectées. En effet, certains d'entre eux étaient soucieux que les résultats des tests soient transmis aux supérieurs. Afin de rassurer les participants de l'expérience, nous avons

décidé de rendre anonyme les données et d'attribuer un numéro de sujet à chaque participant. Cette décision correspond à la recommandation de confidentialité de Wholin et al. énoncée en Section 6.3.4.

CHAPITRE 12

Analyse de la performance

APRÈS le déroulement de l'expérience, les résultats peuvent être analysés et interprétés. La première approche que nous abordons, au regard de la Section 5.2, est une analyse de la performance de la tâche primaire. Cette analyse fait partie des trois analyses effectuées avec les données récoltées lors de l'expérience. Toutes les analyses ont été effectuées à l'aide du logiciel ***R****[R D10] et suivent strictement la méthode présentée dans [WRH+99].*

Ce chapitre se présente sous deux axes : l'analyse des variables indépendantes (l'influence du patron architectural et du framework sur les variables dépendantes que sont le temps et le caractère d'exactitude des réponses) et, d'autre part, l'analyse des facteurs secondaires (sexe et niveau d'étude) sur les variables dépendantes étudiées.

Rappelons ici que, comme indiqué en Section 10.4.4, les sujets retenus pour cette analyse sont ceux ayant correctement effectué les tâches de maintenance.

Sommaire

12.1 Approche et mise en place

Deux variables sont mesurées pour l'évaluation de la performance de la tâche principale (voir Section 5.2) : le temps passé sur chaque tâche et le caractère d'exactitude des réponses. Afin de mesurer le temps passé sur chaque question, nous utilisons les données fournies par l'oculomètre. En effet, avant de répondre à une question, chaque sujet appuie sur une touche du clavier afin d'afficher le diagramme. Le sujet appuie également sur une touche du clavier lorsqu'il pense avoir trouvé la réponse (comme décrit en Section 11.3.1). Ces données, une fois importées dans TAUPE, nous permettent facilement de connaître le temps exact passé sur chaque tâche par chaque sujet.

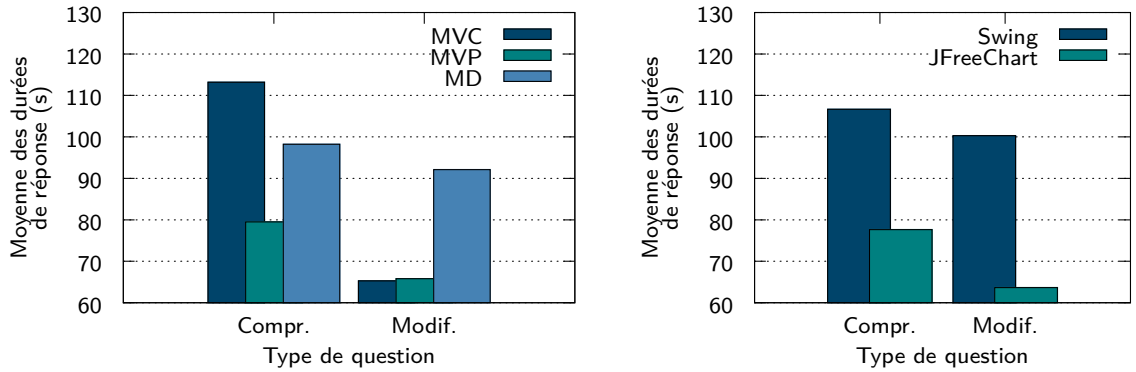
Au sujet de l'exactitude des réponses, une fois que le sujet a terminé de répondre aux différentes questions, a remplit le questionnaire post-expérimental (voir Section 10.8) et qu'il nous a rendu le questionnaire, nous procédons à la correction des réponses et nous encodons le fait qu'une réponse soit correcte ou non dans un fichier pouvant être importé dans TAUPE (pour plus de détail, se référer au guide de l'utilisateur fourni en Annexe D). Les résultats concernant l'exactitude des réponses sont directement calculés par TAUPE.

12.2 Étude de l'influence des variables indépendantes : temps passé sur une tâche

L'étude du temps passé sur une tâche peut sembler à priori un bon indicateur de la charge de travail nécessaire pour accomplir cette même tâche. Cependant, il est nécessaire de comprendre le fait que deux sujets peuvent passer le même temps sur un diagramme mais ne pas utiliser ce temps de manière optimale. Alors que l'un peut passer son temps à analyser les mauvaises classes pour finir par trouver les classes pertinentes et réaliser la tâche rapidement, un autre peut passer tout son temps dans les classes pertinentes et avoir de réelles difficultés à effectuer la tâche. C'est cet élément qui justifie en majorité l'utilisation d'un oculomètre.

12.2.1 Hypothèses

Les différentes hypothèses étudiées dans cette section sont explicitées ci-dessous. Celles-ci tentent d'établir un lien entre les variables indépendantes (patron architectural et *framework* utilisé) et la variable dépendante qu'est le **temps passé à effectuer une tâche**.



(a) Selon le patron

(b) Selon le framework

FIGURE 12.1 – Étude des moyennes des temps de réalisation des tâches

H_{0a}	Les tâches de maintenance nécessitent, en moyenne, le même temps quelle que soit la variante X du patron architectural utilisée.
H_{0b}	Les tâches de maintenance nécessitent, en moyenne, le même temps quelle que soit la variante Y du *framework* utilisée.
H_{0c}	Les tâches de maintenance nécessitent, en moyenne, le même temps quelle que soit la variante X du patron architectural utilisée et quel que soit le *framework* Y utilisé.
H_{1a}	La variante X nécessite, en moyenne, un temps significativement plus faible que les autres pour les tâches de maintenance.
H_{1b}	Le *framework* Y, nécessite, en moyenne un temps significativement plus faible que les autres pour les tâches de maintenance.
H_{1c}	La variante X, nécessite, un temps significativement plus faible que les autres pour les tâches de maintenance quel que soit le *framework* Y utilisé.
	$X \in \{\text{MVC}, \text{MD}, \text{MVP}\}$
	$Y \in \{\text{JFreeChart}, \text{Swing}\}$

12.2.2 Données

Les Tableaux A.1 et A.2 présentent les moyennes des temps de réalisation des tâches de maintenance selon les variables indépendantes. Le Graphique 12.1a illustre ces données en fonction du patron architectural et le Graphique 12.1b illustre la moyenne en fonction du *framework*.

12.2.3 Test d'hypothèse

Le Tableau 12.1 présente les tests d'hypothèses ANOVA pour les hypothèses présentées en Section 12.1. Nous remarquons que le seul résultat significatif concerne le *framework* (p-value $=$ 0.003663). Une *p-value* inférieure au seuil $\alpha = 0.01$ permet de conclure au rejet d'hypothèse nulle comme détaillé en Section 7.2.5. La seule hypothèse nulle qu'il est donc possible de rejeter est l'hypothèse H_{0b} (avec une probabilité significative $\alpha = 0.01$). Les autres *p-values* étant non significatives, c'est la seule conclusion qu'il est possible de tirer. Par conséquent, nous pouvons affirmer (avec une probabilité d'erreur inférieure à 0.01) que :

	Patron	Framework	Patron/Framework	Residu
Degré de liberté	2	1	2	96
Somme des racines	7499	37140	2808	401784
Carré des moyennes	3750	37140	1404	4185
F-Value	0.8959	8.8741	0.3355	
P-Value	0.411629	0.003663	0.715849	

TABLE 12.1 – Test d'hypothèse ANOVA pour le temps passé sur les tâches de maintenance

	Swing	JFreeChart
Modification	0.3185	0.3492
Compréhension	0.5027	0.446

TABLE 12.2 – *p-values* de l'influence du patron architectural sur le temps par *framework* en fonction du type de tâche

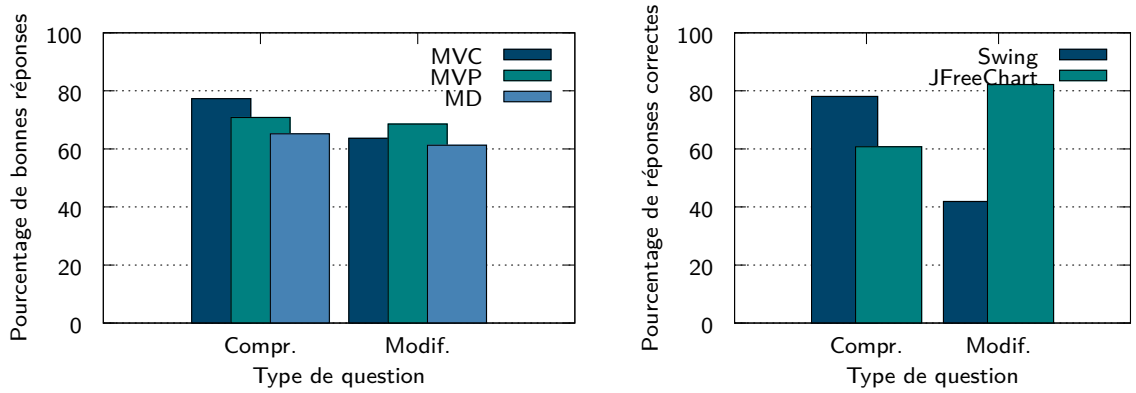
Conclusion 1. ***Le framework JFreeChart nécessite, en moyenne, un temps significativement plus faible que les autres pour les tâches de maintenance.***

Le rejet de cette hypothèse implique que pour la suite des analyses concernant le temps, il sera nécessaire de bien distinguer les deux *frameworks* utilisés.

D'autres analyses ont également été effectuées telles que l'analyse de l'impact du patron architectural sur le type de tâche effectuée (compréhension ou modification), et ce pour chacun des deux *frameworks* étudiés (*JFreeChart* et *Swing*). Néanmoins, étant donné qu'aucune de ces données n'a fourni de résultat significatif, ces analyses ne sont pas présentées dans ce document. Le Tableau 12.2 présente les *p-values* obtenues lors de ces analyses à l'aide du test Kruskal-Wallis.

12.3 Étude de l'influence des variables indépendantes : validité des réponses

L'exactitude des réponses est étudiée afin de mesurer la *difficulté* associée à chaque variable indépendante. Le fait d'effectuer une tâche correctement est un élément essentiel de notre analyse. En effet, cette variable permet non seulement d'éliminer les réponses des sujets ayant mal réalisé une tâche (les différentes mesures perdant donc leur sens) mais également d'estimer si tel ou tel patron architectural offre un plus grand taux de réussite. Si toutes les analyses effectuées dans ce document n'utilisent que les réponses correctes aux tâches de maintenance, il est évident que l'analyse de l'exactitude des réponses considère toutes les réponses fournies.



(a) Selon le patron

(b) Selon le framework

FIGURE 12.2 – Exactitude des réponses aux tâches de maintenance

12.3.1 Hypothèses

H_{0a}	En moyenne, le taux de réponses exactes est le même, quelle que soit la variante X du patron architectural utilisée.
H_{0b}	En moyenne, le taux de réponses exactes est le même, quelle que soit la variante Y du *framework* utilisée.
H_{0c}	En moyenne, le taux de réponses exactes est le même, quelle que soit la variante X du patron architectural utilisée et quel que soit le *framework* Y utilisé.
H_{1a}	La variante X implique, en moyenne, un taux de réponses exactes plus élevé que les autres pour les tâches de maintenance.
H_{1b}	Le *framework* Y implique, en moyenne, un taux de réponses exactes plus élevé que les autres pour les tâches de maintenance.
H_{1c}	La variante X implique, en moyenne, un taux de réponses exactes plus élevé que les autres pour les tâches de maintenance quel que soit le *framework* Y utilisé.
	$X \in \{\text{MVC}, \text{MD}, \text{MVP}\}$
	$Y \in \{\text{JFreeChart}, \text{Swing}\}$

12.3.2 Données

Le Tableau A.3 et le Tableau A.4 présentent respectivement les taux de bonnes réponses aux tâches de maintenance selon le patron architectural et selon le *framework*. Le Graphique 12.2a illustre ces données en fonction du patron architectural et le Graphique 12.2b illustre la moyenne en fonction du *framework*.

12.3.3 Test d'hypothèse

Le Tableau 12.3 présente le test d'hypothèse ANOVA appliqué à l'exactitude des réponses fournies par les sujets en fonction du patron architectural et du *framework* utilisé. Nous remarquons que la seule *p-value* capable de rejeter une hypothèse est celle du *framework*. Effectivement, celle-ci est significative pour un $\alpha = 0.05$. Nous pouvons donc rejeter l'hypothèse H_{0b} et valider l'hypothèse

	Framework	Patron	Framework/Patron	Residu
Degré de liberté	1	2	2	162
Somme des carrés	1.006	0.097	0.532	35.358
Moyenne des carrés	1.00595	0.04865	0.26620	0.21826
F-value	4.6089	0.2229	1.2196	
P-Value	0.03329	0.80044	0.29803	

TABLE 12.3 – Test d'hypothèse ANOVA pour la validité des réponses

	Swing	JFreeChart
Modification	0,5206	0,9003
Compréhension	0,701	0,05859

TABLE 12.4 – *p-values* de l'influence du patron architectural sur le taux de bonnes réponses par *framework* en fonction du type de tâche

H_{1b}. Au vu des autres *p-values*, il s'agit de la seule conclusion possible. Nous pouvons donc affirmer (avec une probabilité d'erreur inférieure à 0.05) que :

Conclusion 2. ***Le framework JFreeChart implique, en moyenne, un taux de réponses exactes plus élevé que les autres pour les tâches de maintenance.***

Étant donné que le *framework* utilisé influence le taux de réponses correctes, nous effectuons une analyse plus avancée afin de déterminer si des hypothèses équivalentes peuvent être prouvées pour chacun des deux *frameworks*. Nous séparons les données des *frameworks* afin d'étudier l'influence des patrons architecturaux sur le taux de réponses correctes, et ce en fonction du type de tâche à effectuer (compréhension ou modification). Ces analyses n'ayant pas donné de résultats significatifs (pour un $\alpha = 0.05$) pour le *framework* Swing, nous ne présentons pas cette analyse ici. Les résultats des *p-values* calculées selon le test Kruskal-Wallis sont néanmoins présentées dans le Tableau 12.4.

12.4 Étude de l'influence du sexe des sujets

Après avoir analysé et mis en évidence que seuls les *frameworks* étudiés ont une influence sur le temps de réponse et le taux de réponses correctes, nous nous interrogeons sur l'influence du facteur second qu'est le **sexe** sur les variables dépendantes étudiées dans cette section : le temps de réponse et la validité des réponses. Au vu des résultats précédemment obtenus, nous savons qu'il est nécessaire de bien distinguer les deux *frameworks* utilisés.

Nous ne présentons ici que les tests nous ayant permis de rejeter ou d'accepter certaines hypothèses. Par conséquent, l'influence du sexe sur les temps de réalisation de tâches n'est pas présentée.

12.4.1 Influence du sexe sur la validité des réponses : Swing

Nous sommes en droit de nous demander s'il existe un lien entre le taux de bonnes réponses des sujets et le sexe de ceux-ci. Cette section détaille l'analyse de ce lien.

W	P-value
729.5	0.1237

TABLE 12.5 – Test d'hypothèse Mann-Whitney pour le taux de bonnes réponses en fonction du sexe du sujet pour le *framework* Swing

Hypothèses

H_{0a}	Pour Swing, en moyenne, le taux de réponses exactes est le même, quel que soit le sexe du sujet.
H_{1a}	Pour Swing, en moyenne, le taux de réponses exactes est plus élevé pour un sexe que pour l'autre.

Données

Les moyennes de taux de réponses correctes pour les hommes et les femmes sont respectivement de 0.76 et 0,59.

Test d'hypothèse

Le Tableau 12.5 présente le test d'hypothèse Mann-Whitney sur les données. La *p-value* qui ressort montre que les résultats ne sont pas significatifs et qu'il n'est donc pas possible de conclure quoi que ce soit sur l'influence du sexe sur la validité des réponses pour Swing.

12.4.2 Influence du sexe sur la validité des réponses : JFreeChart

Une fois le *framework* Swing analysé, il est nécessaire d'appliquer la même analyse pour le *framework* JFreeChart.

Hypothèses

H_{0a}	Pour JFreeChart, en moyenne, le taux de réponses exactes est le même, quel que soit le sexe du sujet.
H_{1a}	Pour JFreeChart, en moyenne, le taux de réponses exactes est plus élevé pour un sexe que pour l'autre.

Données

Les moyennes de taux de réponses correctes pour les hommes et les femmes sont respectivement de 0,74 et de 0,75.

Test d'hypothèse

Étant donné que les moyennes entre hommes et femmes sont exactement identiques, un test d'hypothèse n'est pas nécessaire, il est directement possible de conclure que nous pouvons rejeter l'hypothèse H_{1a} avec un $\alpha = 0$ et donc affirmer H_{0a}.

Conclusion 3. ***Pour JFreeChart, en moyenne, le taux de réponses exactes est le même, quel que soit le sexe du sujet.***

Pour un des deux *frameworks*, le sexe n'a aucune influence sur le taux de réponse correct (Conclusion 3) alors qu'il n'est pas possible d'affirmer quoi que ce soit sur Swing. Nous pouvons raisonnablement conclure, avec une probabilité d'erreur inférieure à 0.05, que :

Conclusion 4. ***En moyenne, le taux de réponses exactes est le même, quel que soit le sexe du sujet.***

12.5 Étude de l'influence du niveau d'étude des sujets

La seconde caractéristique des sujets qui pourrait influencer les résultats est le niveau d'étude des sujets. Cette section présente les différents tests d'hypothèses effectués afin de montrer le lien entre le niveau d'étude et le temps de réponse, mais également le lien entre le niveau d'étude et le taux de bonnes réponses. Les analyses ont été effectuées séparément pour chaque *framework*. Seules les analyses ayant montré des résultats significatifs sont présentées dans cette section.

Notons que la caractéristique "Master" ou "Doctorant" est liée au niveau d'étude obtenu **ou en cours**. Dans la suite de cette section, nous appelons donc "Master" toute personne ayant comme diplôme le plus élevé un master, ou toute personne effectuant des études en vue de l'obtention de ce diplôme. Nous appelons "Doctorant", toute personne ayant comme diplôme le plus élevé un doctorat, ou toute personne effectuant des études en vue de l'obtention de ce diplôme.

12.5.1 Influence du niveau d'étude sur le temps de réponse : Swing

Hypothèses

H_{0a}	Pour Swing, les tâches de maintenance nécessitent, en moyenne, le même temps quel que soit le niveau d'étude du sujet.
H_{1a}	Pour Swing, les tâches de maintenance nécessitent, en moyenne, moins de temps pour un certain niveau d'étude.

Données

Les moyennes de temps nécessaire pour répondre aux tâches en fonction du niveau d'étude sont de 143,77 secondes pour les Doctorants et de 79,24 secondes pour les Masters. Les Graphiques 12.3 et 12.4 présentent ces données sous forme graphique.

Test d'hypothèse

Le Tableau 12.6 présente le test d'hypothèse Mann-Whitney sur les données. Comme la Figure 12.3 peut le laisser entrevoir, les Masters prennent significativement moins de temps pour répondre aux tâches de maintenance que les Doctorants sur le *framework* Swing. Cette intuition est confirmée par la *p-value* du Tableau 12.6. Au vu de ces données, il est possible de rejeter l'hypothèse H_{0a} et d'affirmer, avec une probabilité d'erreur inférieure à 0.01, H_{1a} :

Conclusion 5. ***Pour Swing, les tâches de maintenance nécessitent, en moyenne, moins de temps pour les sujets Master que pour les sujets Doctorants.***

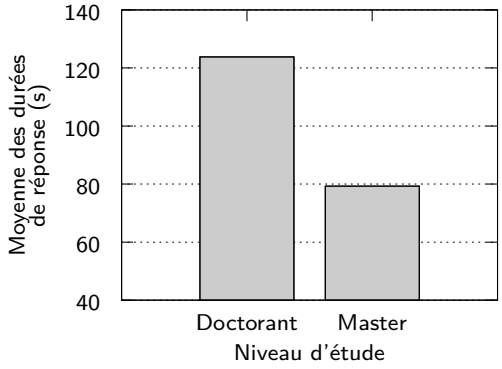


FIGURE 12.3 – Moyennes de temps mis pour réaliser les tâches en fonction du niveau d'étude du sujet (Swing)

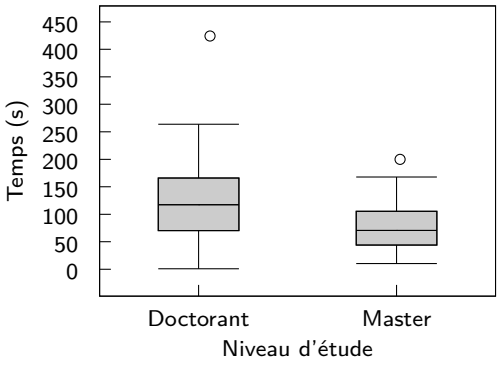


FIGURE 12.4 – Boîte à moustaches des données montrant l'influence du niveau d'étude sur le temps mis pour réaliser les tâches (Swing)

W	P-value
346	0.04763

TABLE 12.6 – Test d'hypothèse Mann-Whitney pour le temps passé sur les tâches de maintenance en fonction du niveau d'étude pour Swing

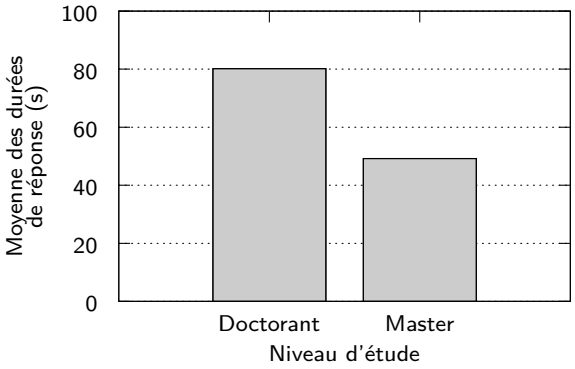


FIGURE 12.5 – Moyennes de temps mis pour réaliser les tâches en fonction du niveau d'étude du sujet (JFreeChart)

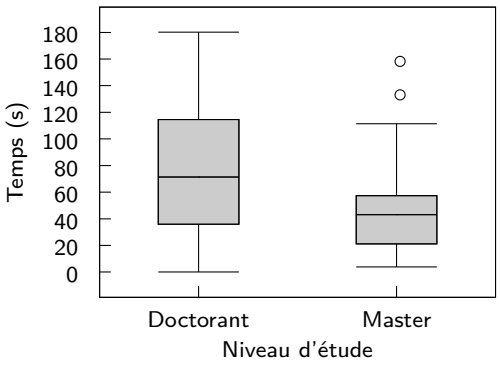


FIGURE 12.6 – Boîte à moustaches des données montrant l'influence du niveau d'étude sur le temps mis pour réaliser les tâches (JFreeChart)

12.5.2 Influence du niveau d'étude sur le temps de réponse : JFreeChart

Hypothèses

H_{0a}	Pour JFreeChart, les tâches de maintenance nécessitent, en moyenne, le même temps quel que soit le niveau d'étude du sujet.
H_{1a}	Pour JFreeChart, les tâches de maintenance nécessitent, en moyenne, moins de temps pour un certain niveau d'étude.

Données

Les moyennes de temps nécessaire pour répondre aux tâches en fonction du niveau d'étude sont de 93,92 secondes pour les Doctorants et de 49,14 secondes pour les Masters. Les Graphiques 12.5 et 12.6 présentent ces données sous forme graphique.

W	P-value
481	0.09271

TABLE 12.7 – Test d'hypothèse Mann-Whitney pour le temps passé sur les tâches de maintenance en fonction du niveau d'étude pour JFreeChart

Test d'hypothèse

Le Tableau 12.7 présente le test d'hypothèse Mann-Whitney sur les données. Comme la Figure 12.5 peut le laisser entrevoir, les Masters prennent significativement moins de temps pour répondre aux tâches de maintenance que les Doctorants sur le *framework* JFreeChart. Cette intuition est confirmée par la *p-value* du Tableau 12.7. Au vu de ces données, il est possible de rejeter l'hypothèse H_{0a} et d'affirmer, avec une probabilité d'erreur inférieure à 0.1, H_{1a} :

Conclusion 6. ***Pour JFreeChart, les tâches de maintenance nécessitent, en moyenne, moins de temps pour les sujets Master que pour les sujets Doctorants.***

Au vu des Conclusions 5 et 6, et étant donné que le patron architectural n'influence rien, nous pouvons affirmer avec une probabilité d'erreur inférieure à $0.05 \sim 0.1$ que :

Conclusion 7. ***En moyenne, les étudiants Master prennent moins de temps pour répondre aux questions que les sujets Doctorants.***

12.6 Conclusions

L'analyse des données concernant les variables indépendantes que sont le patron architectural et le *framework* nous permettent d'établir trivialement que le *framework* influence de manière significative le temps de réponse. Nous avons également montré que le sexe n'a aucune influence significative sur le temps de réponse (Conclusion 4) et que les Master [1] prennent en moyenne moins de temps que les Doctorants [2] pour effectuer les tâches de maintenance (Conclusion 7). Cependant, aucune conclusion n'a pu être tirée du point de vue de la qualité de leurs réponses.

1. Rappelons que nous appelons "Master" toute personne ayant comme diplôme le plus élevé un Master, ou toute personne effectuant des études en vue de l'obtention de ce diplôme

2. Nous appelons "Doctorant", toute personne ayant comme diplôme le plus élevé un doctorat, ou toute personne effectuant des études en vue de l'obtention de ce diplôme

CHAPITRE 13

Analyse des mesures physiologiques

La deuxième partie de notre analyse des données collectées durant notre expérience concerne les mesures physiologiques, à savoir, dans notre cas, les mesures oculométriques. La première partie pose le contexte et l'approche adoptée pour l'analyse. La deuxième partie résume l'analyse des données oculométriques sous forme de tableaux de p-values. Ces tableaux permettent de rapidement cerner quelles analyses sont les plus pertinentes. La troisième concerne l'analyse des données oculométriques vis-à-vis des variables indépendantes "patron architectural" et "framework". La quatrième et la cinquième partie traitent de l'influence de facteurs confondants que sont le sexe et le niveau d'étude des sujets. Pour finir, la dernière partie conclut l'analyse des mesures physiologiques.

Rappelons ici que, comme indiqué en Section 10.4.4, à cause de problèmes rencontrés durant les expériences (Section 11.3.2) certains enregistrements sont partiellement pris en compte voire pas du tout. Par conséquent, certaines données pour certaines questions n'ont pas été prises en compte pour cette analyse (ces données n'étant pas du tout exploitables).

Sommaire

13.1 Approche et mise en place

Afin de ne pas baser uniquement notre analyse sur la performance (analysée dans le Chapitre 12), nous étudions également l'influence des différentes variables indépendantes prises en compte sur les variables dépendantes calculées sur base des informations fournies par l'oculomètre. Comme expliqué en Section 8.2, le logiciel TAUPE nous permet d'importer les données fournies par les oculomètres afin de générer des fichiers `.csv` directement exploitables dans un logiciel d'analyse de données tel que **R**.

De nombreuses variables oculométriques issues de la littérature et des expériences précédentes (Section 4.2.3) ont été calculées à l'aide de TAUPE. Ces différentes variables sont détaillées en Section 4.2.3. Le but de l'analyse des données oculométriques est d'ajouter un niveau de granularité plus fin dans l'analyse de l'effort fourni pour effectuer les tâches de maintenance. En effet, ces données nous permettent de savoir exactement où le sujet a posé son regard (dans des classes pertinentes ou non) et donc de pouvoir estimer la pertinence de son effort.

La totalité des variables étudiées ici sont formellement définies en Section 4.2.3. Le choix des variables étudiées est justifié ci-dessous :

Densité spatiale La densité spatiale est retenue comme variable d'analyse car elle nous permet d'évaluer la "surface" couverte par les fixations du sujet. Plus la densité est faible, plus la recherche est dirigée, peu importe l'ordre dans lequel les fixations ont été enregistrées [GK99].

Densité de transition La densité de transition est retenue car elle est l'équivalent "dynamique" de la densité spatiale où l'ordre des fixations est déterminant.

IN AORI / IN AOII normalisé : Le taux normalisé de fixations est pris en compte car il permet d'évaluer le taux de fixations réellement pertinentes. Cette métrique reprend le *IN AORI / IN AOII* et le principe du taux normalisé de fixations par zone d'intérêt.

Durée moyenne des fixations La durée des fixations est présentée car, en théorie, elle permet de savoir, en moyenne, combien de temps le sujet a fixé un endroit précis du diagramme. Plus ce temps est élevé, plus le sujet a dû se concentrer sur le diagramme. Cependant, bien que les résultats de l'analyse de ces données soient significatifs (Tableaux 13.1 et 13.2), nous considérons ces données comme non pertinentes. Cette décision est basée sur les observations faites durant la surveillance de l'expérience. En effet, nous avons remarqué que les temps de fixations avaient tendance à varier très fortement d'un sujet à l'autre, voire même d'un diagramme à l'autre pour un même sujet. Ces variations étant dues à des faiblesses de l'oculomètre. De ce fait, nous présentons les résultats en Section 13.2 mais l'analyse détaillée des données n'est pas présentée et aucune conclusion ne sera tirée sur base de ces données.

Enveloppe convexe : La surface de l'enveloppe convexe est une mesure de la pertinence de la recherche. Nous avons décidé de garder 60% des fixations les plus longues des sujets afin d'éliminer les fixations de courte durée peu pertinentes. La surface de l'enveloppe convexe des points restants est un bon indicateur de l'effort visuel requis pour réaliser la tâche. Une petite surface induit une recherche localisée et précise alors qu'une grande surface induit une recherche plus fastidieuse. En l'occurrence, l'utilisation de cette métrique est pertinente au vu de la similarité visuelle des différents diagrammes d'un même *framework*.

Moyenne des distances relatives entre chemins visuels : Nous présentons l'analyse de la métrique dont nous sommes les auteurs afin de pouvoir l'appliquer sur un cas concret.

	Patron	Framework
Densité spatiale	0,9833	0,00217
Densité de transition	0,373	0,0009629
IN AORI / IN AOII normalisé	0,3363	0,02811
Durée moyenne des fixations	0,3874	0.0165
Enveloppe Convexe	0.2913	0.08416

TABLE 13.1 – *P-values* des différentes variables dépendantes pour l'analyse oculométrique selon les variables indépendantes

L'analyse des facteurs confondants que sont le sexe et le niveau d'étude sont des analyses complémentaires, non directement liées au sujet traité dans ce document mais, étant donné que nous avons accès à ces informations, nous avons jugé intéressant de les traiter. Pour la totalité des analyses, nous avons fixé la valeur α à 0.05 comme il est commun de le faire.

13.2 Résumé des analyses

13.2.1 Analyse des métriques basées sur les fixations

Cette section résume les différentes analyses et tests d'hypothèses effectués sur les **métriques basées sur les fixations**. Les *p-values* (voir Section 7.2.5) présentées ici permettent d'évaluer l'influence des variables indépendantes (patron et *framework*) sur les variables dépendantes. Une *p-value* inférieure à $\alpha = 0.05$ permet de confirmer l'influence de la variable indépendante sur la variable dépendante. Ces valeurs sont présentées sur fond vert dans les tableaux.

Nous avons également choisi de mettre en avant les *p-values* comprises entre 0.05 et 0.1 étant donné le fait qu'elles sont "presque significatives" pour l'alpha que nous avons fixé. Néanmoins, nous ne considérerons pas qu'elles permettent de confirmer ou de rejeter une hypothèse. Ces valeurs sur fond orange sont présentées à titre indicatif.

Notons également qu'une valeur de $(1 - \text{pvalue})$ inférieure à α permet de conclure à la non-influence d'une variable indépendante sur une variable dépendante. Toutes les *p-values* ont été calculées via la méthode **ANOVA** (décrite en Section 7.2). Le choix de la méthode d'analyse a été effectué selon Wohlin et al. [WRH+99]. Tous les tests d'hypothèses ont été réalisés à l'aide du logiciel **R**.

Il est relativement aisé de voir dans le Tableau 13.1, que la seule conclusion possible concernant l'influence du patron architectural est sa **non-influence sur la densité spatiale** (*p-value* de 0.9833). Cette analyse est effectuée en détail en Section 13.3.

Il est également remarquable que toutes les variables dépendantes semblent être influencées par le *framework* utilisé (troisième colonne du Tableau 13.1). Cette influence est **tout à fait normale** étant donné que le *framework* implique un diagramme significativement différent. Cette conclusion nous permet tout de même de vérifier que les données que nous avons récoltées sont bien pertinentes et que les prédictions faites sur les métriques se confirment au moment de l'analyse. Il est possible de conclure que :

Conclusion 8. ***Les variables : Densité spatiale, densité de transition, IN AORI / IN AOII normalisé sont toutes significativement influencées par le framework utilisé.***

	Sexe	Niveau d'étude
Densité spatiale	0.3522	0,01009
Densité de transition	0.2552	0.09749
IN AORI / IN AOII normalisé	0.6332	0.1372
Durée moyenne des fixations	0.01060	0.07135
Enveloppe convexe	0,809	0.004962

TABLE 13.2 – *P-values* de l'influence des facteurs confondants sur les variables dépendantes pour l'analyse oculométrique

Nous n'analysons pas plus en détail la direction de cette influence étant donné que le *framework* n'est pas l'objet d'étude du présent document et que cette analyse n'apporterait aucune information intéressante. En effet, cette analyse est obligatoire et ses résultats sont triviaux. Ils permettent de vérifier la pertinence de la conception de l'expérience. Par exemple, il est normal qu'un sujet ne regarde pas le diagramme JFreeChart de la même façon que le diagramme Swing étant donné qu'ils sont totalement différents. Nous n'analysons pas non plus en détail l'influence des variables indépendantes sur la densité de transition même s'il est intéressant de remarquer que celle-ci semble être également influencée par le *framework* bien que l'alpha utilisé afin de confirmer cette hypothèse soit légèrement supérieur à 0,05.

13.2.2 Analyse des facteurs confondants

Cette section résume les différents résultats des analyses effectuées sur l'**influence du sexe et du niveau d'étude sur les variables dépendantes**. Bien que ces analyses ne soient pas le but premier de ce document, les caractéristiques des sujets ayant passé notre expérience (Section 10.4) et le fait que la répartition homme / femme soit homogène nous ont fourni des données que nous avons estimé suffisamment précieuses pour les analyser brièvement. Toutes les *p-values* ont été calculées via le test Mann-Whitney, déterminée grâce à [WRH+99]. Tous les calculs ont été effectués par le logiciel **R**.

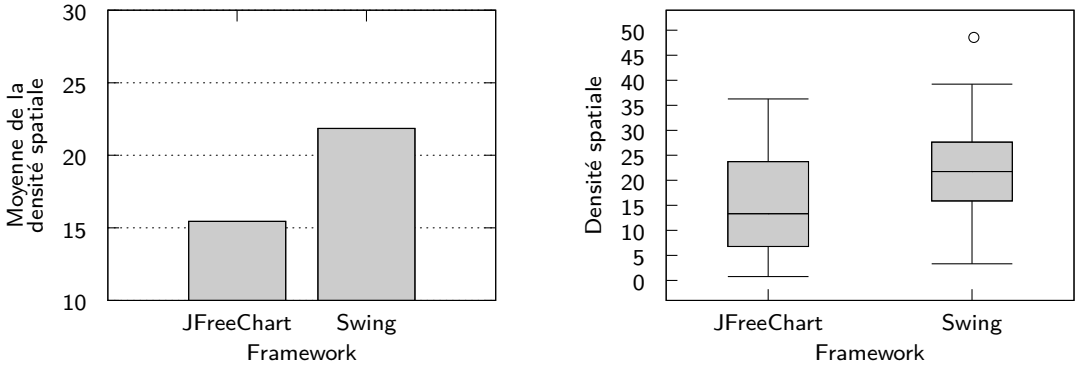
Concernant le sexe des sujets, la seule variable influencée par ce facteur est la durée moyenne des fixations. Toutefois, comme il est précisé en Section 13.1, cette métrique n'est pas retenue comme étant pertinente. De ce fait, **aucune influence significative du sexe sur les variables oculométriques n'a été détectée**.

Concernant le niveau d'étude des sujets, il est notable que quatre des cinq *p-values* sont en dessous d'un $\alpha = 0.1$, ce qui correspond à une probabilité inférieure à 0.1 de faire une erreur de première espèce, c'est-à-dire d'obtenir un faux négatif. Nous pouvons raisonnablement penser que le niveau d'étude influe de manière significative sur les différentes variables oculométriques. L'analyse de l'influence du niveau d'étude sur la densité de transition est présentée en détail en Section 13.4. Toutes les autres analyses sont semblables à celle-ci et peuvent être facilement déduites de l'analyse de la densité de transition. Notons que les conclusions sont les mêmes, les Masters ont des données oculométriques qui tendent à montrer que leur pertinence de recherche est plus élevée que pour les Doctorants. De ce fait, ces analyses ne sont pas présentées en détail dans ce document. Néanmoins, il est possible de conclure :

Conclusion 9. ***La Densité spatiale est significativement plus faible pour les Masters que pour les Doctorants.***

	Sexe	Niveau d'étude	Type de tâche
DRCV	0.2445	2.463e-08	4.472e-16

TABLE 13.3 – *P-values* des distances relatives entre chemins visuels



(a) Moyenne (b) Boîte à moustaches

FIGURE 13.1 – Étude de la densité spatiale en fonction du *framework*

13.2.3 Analyse de la Distance Relative entre Chemins Visuels

La métrique que nous proposons dans ce document, la *Distance Relative entre Chemins Visuels*, est appliquée aux données que nous avons récoltées. Ces données portant sur les moyennes des différences entre chemins visuels d'un groupe, leurs analyses ne portent pas sur les diagrammes eux-mêmes, mais tentent de mettre en avant les différences (si elles existent), dans la façon de parcourir ces diagrammes. Cette métrique est donc appliquée entre les hommes, les femmes, les Masters et les Doctorants pour chacune des questions de notre expérience. L'analyse présentée dans cette section compare les moyennes entre les différentes questions.

Le Tableau 13.3 présente les différentes *p-values* concernant la *Distance Relative entre Chemins Visuels* obtenues à l'aide du test Mann-Whitney. Ce tableau permet d'avoir un rapide aperçu des analyses donnant des résultats significatifs. Nous remarquons aisément que la distance relative entre chemins visuels est influencée le niveau d'étude et le type de tâche à effectuer. Le détail de l'analyse de ces différentes *p-values* est présenté Section 13.5. Il n'est pas possible de tirer de conclusion au sujet du sexe.

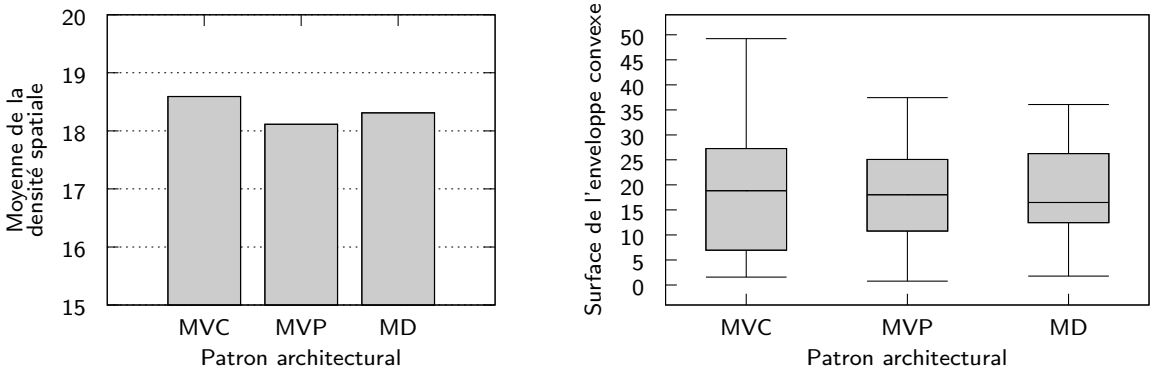
13.3 Étude de l'influence des variables indépendantes

Cette section présente l'analyse détaillée de l'influence du *framework* et du patron architectural sur la densité spatiale.

13.3.1 Données

Les différentes données utilisées pour cette analyse sont détaillées dans le Tableau A.6. La Figure 13.1a présente un graphique des moyennes des densités spatiales par *framework* et la Figure 13.1 présente la répartition des données de la densité spatiale, toujours en fonction du *framework*.

Concernant l'influence du patron architectural, la Figure 13.2a présente les moyennes de densité spatiale en fonction du patron architectural et la Figure 13.2 illustre la répartition des données pour



(a) Moyenne

(b) Boîte à moustaches

FIGURE 13.2 – Étude de la densité spatiale en fonction du patron architectural

cette métrique.

13.3.2 Hypothèses

H_{0a}	La densité spatiale est, en moyenne, la même quelle que soit la variante X du patron architectural utilisée.
H_{0b}	La densité spatiale est, en moyenne, la même quelle que soit la variante Y du *framework* utilisée.
H_{0c}	La densité spatiale est, en moyenne, la même quelle que soit la variante X du patron architectural utilisée et quel que soit le *framework* Y utilisé.
H_{1a}	La variante X a, en moyenne, une densité spatiale plus faible.
H_{1b}	La variante Y a, en moyenne, une densité spatiale plus faible.
H_{1c}	La variante X a, en moyenne, une densité spatiale plus faible quel que soit le *framework* Y utilisé.
	$X \in \{\text{MVC}, \text{MD}, \text{MVP}\}$
	$Y \in \{\text{JFreeChart}, \text{Swing}\}$

13.3.3 Test d'hypothèse

Le Tableau 13.4 présente le test d'hypothèse ANOVA appliqué aux données. Comme les figures présentées dans la section précédente peuvent le suggérer, il existe bien une influence significative du *framework* sur la densité spatiale et une non influence du patron architectural sur cette même densité.

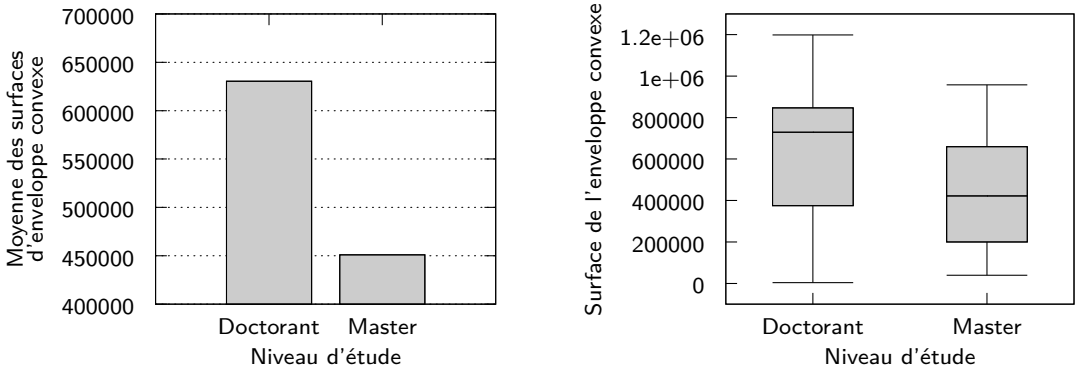
Plus formellement, la *p-value* associée au test de l'influence du *framework* (0.002202) nous permet de rejeter l'hypothèse H_{0b} et d'affirmer, avec une probabilité d'erreur inférieure à 0.05, H_{1b} :

Conclusion 10. ***Le framework JFreeChart a une densité spatiale significativement plus faible que Swing.***

Également, la *p-value* associée à l'influence du patron architectural sur la densité spatiale (0.981976) nous permet de rejeter H_{1a} et d'affirmer, avec une probabilité d'erreur inférieure à 0.05, H_{0a} :

	Patron	Framework	Patron/Framework	Résidu
Degré de liberté	2	1	2	94
Somme des carrés	3.8	1044.3	105.1	9904.2
Carré des moyennes	1.92	1044.3	52.55	105.36
F-Value	0.0182	9.9113	0.4988	
Pr(>F)	0.981976	0.002202	0.608868	

TABLE 13.4 – Test d'hypothèse de l'influence du patron architectural et du *framework* sur la densité spatiale



(a) Moyenne

(b) Boîte à moustaches

FIGURE 13.3 – Étude de la surface de l'enveloppe convexe en fonction du niveau d'étude

Conclusion 11. ***Le patron architectural utilisé n'a pas d'influence significative sur la densité spatiale.***

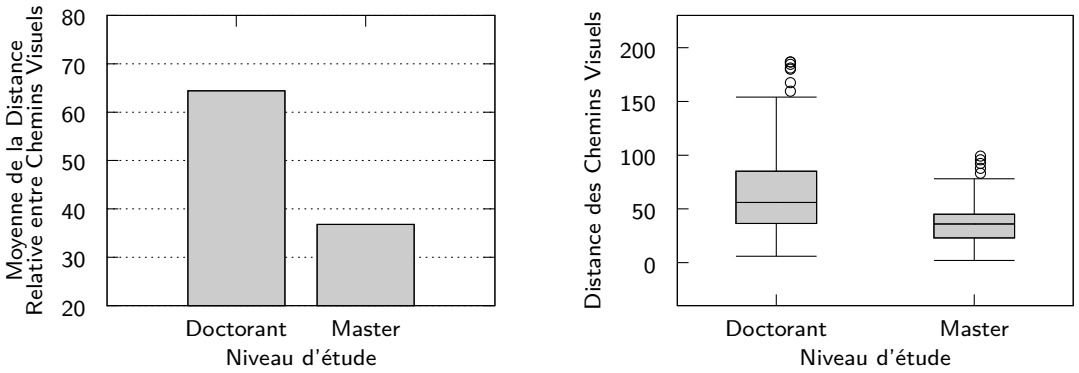
13.4 Étude de l'influence du niveau d'étude des sujets

Étant donné qu'aucune influence du sexe sur les variables oculométriques n'est retenue (l'influence de la durée des fixations n'ayant pas été jugée pertinente comme précisé en Section 13.1), nous ne détaillons ici que le test d'hypothèse de l'influence du niveau d'étude sur l'enveloppe convexe. L'analyse de la densité spatiale étant aisément déductible de cette analyse, celle-ci n'est pas effectuée en détail dans ce document. Il est cependant important de préciser que, en moyenne, la densité spatiale est significativement inférieure pour les Masters.

Les *p-values* des tests d'hypothèses concernant les autres métriques n'étant pas inférieures à $\alpha = 0.05$, l'analyse n'est pas détaillée dans ce document.

13.4.1 Données

Les différentes données utilisées pour cette analyse sont détaillées dans le Tableau A.6. La Figure 13.3a présente un graphique des moyennes des surfaces des enveloppes convexes en fonction du niveau d'étude et la Figure 13.3b la répartition des surfaces en fonction de ces même niveaux d'étude.



(a) Moyenne (b) Boîte à moustaches

FIGURE 13.4 – Étude de la *Distance Relative entre Chemins Visuels* en fonction du niveau d'étude

13.4.2 Hypothèses

H_{0a}	La surface de l'enveloppe convexe est, en moyenne, la même quel que soit le niveau d'étude des sujets.
H_{1a}	La surface de l'enveloppe convexe est, en moyenne, plus faible pour un niveau d'étude que pour l'autre.

13.4.3 Test d'hypothèse

Étant donné que nous sommes dans un cas simple d'un seul facteur à deux traitements, le test d'hypothèse approprié est Mann-Whitney [WRH+99]. La *p-value* ressortant de ce test étant égale à 0.004962, il est possible de rejeter l'hypothèse H_{0a} et donc d'affirmer, avec une probabilité d'erreur inférieure à 0.05, H_{1a} :

Conclusion 12. ***La surface de l'enveloppe convexe est, en moyenne, plus faible pour les Masters que pour les Doctorants.***

13.5 Étude de la Distance Relative entre Chemins Visuels

Nous analysons dans cette section l'influence de facteurs confondants sur la *Distance Relative entre Chemins Visuels*, métrique définie formellement en Section 9. Cette section se divise en deux parties : (1) l'influence du niveau d'étude et (2) l'influence du type de tâche. En l'occurrence, le *framework* n'est pas considéré. L'analyse de cette métrique selon le sexe n'est pas présentée car elle n'est pas significative.

13.5.1 Étude de l'influence du niveau d'étude

Données

Les différentes données utilisées pour cette analyse sont illustrées par les Figures 13.4a et 13.4b qui présentent respectivement un graphique des moyennes des distances relatives entre chemins visuels en fonction du niveau d'étude des sujets et la répartition des distances également en fonction du niveau d'étude.

W	p-value
3505	2.463e-08

TABLE 13.5 – Test d'hypothèse Mann-Whitney pour la *Distance Relative entre Chemins Visuels* en fonction du niveau d'étude du sujet

Hypothèses

H_{0a}	La *Distance Relative entre Chemins Visuels* est, en moyenne, la même quel que soit le niveau d'étude des sujets.
H_{1a}	La *Distance Relative entre Chemins Visuels* est, en moyenne, plus faible pour un niveau d'étude que pour l'autre.

Test d'hypothèse

Étant donné que nous sommes dans un cas simple d'un seul facteur à deux traitements (Masters et Doctorants), le test d'hypothèse approprié est Mann-Whithney [WRH+99] (car les données ne suivent pas une distribution normale). Le Tableau 13.5 présente les résultats de ce test. Comme les graphiques présentés en Figure 13.4b et 13.4a le laissaient présager, la *p-value* est faible et largement inférieure à 0.05. La *p-value* ressortant de ce test est, en effet, égale à 2.463e-08. Il est donc possible de rejeter l'hypothèse H_0a avec une probabilité d'erreur inférieure à 0.01 et donc d'affirmer H_{1a} :

Conclusion 13. ***La Distance Relative entre Chemins Visuels est, en moyenne, plus faible pour les Master que pour les Doctorants.***

Néanmoins, comme pour l'analyse précédente, il est important de remarquer l'existence de valeurs extrêmes (visibles sur la Figure 13.4b), valeurs qui influencent donc fortement la moyenne.

13.5.2 Étude de l'influence du type de tâche

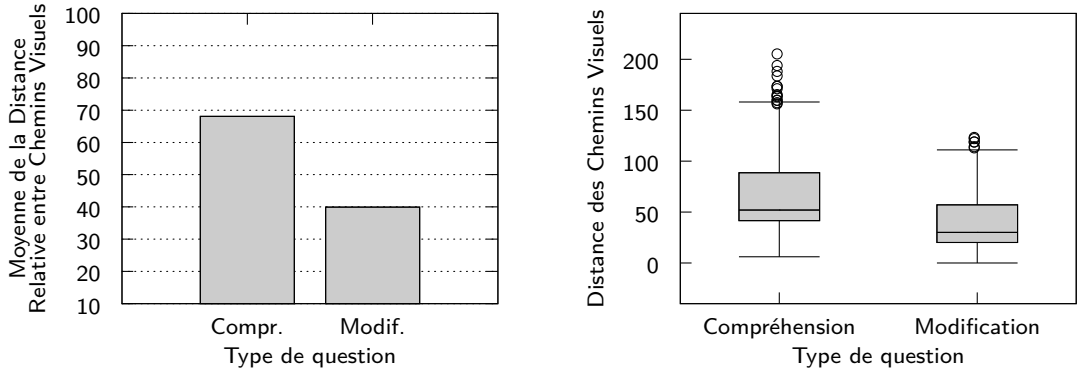
La dernière analyse effectuée sur la distance relative entre chemins visuels est l'analyse de l'influence du type de tâche. En effet, il peut être intéressant d'analyser la différence qu'il existe dans la façon de parcourir un diagramme en fonction du type de tâche à effectuer : une tâche de compréhension ou bien de modification.

Données

Les différentes données utilisées pour cette analyse sont illustrées par les Figures 13.5a et 13.5b qui présentent respectivement un graphique des moyennes des distances relatives entre chemins visuels en fonction du type de tâche effectuée par les sujets et la répartition des distances également en fonction du type de tâche.

Hypothèses

H_{0a}	La *Distance Relative entre Chemins Visuels* est, en moyenne, la même quel que soit le type de tâche effectuée par les sujets.
H_{1a}	La *Distance Relative entre Chemins Visuels* est, en moyenne, plus faible pour un type de tâche que pour l'autre.



(a) Moyenne

(b) Boîte à moustaches

FIGURE 13.5 – Étude de la *Distance Relative entre Chemins Visuels* en fonction du type de tâche

W	p-value
42451.5	4.472e-16

TABLE 13.6 – Test d'hypothèse Mann-Whitney pour la *Distance Relative entre Chemins Visuels* en fonction du type de tâche

Test d'hypothèse

Étant donné que nous sommes dans un cas simple d'un seul facteur à deux traitements (compréhension et modification), le test d'hypothèse approprié est Mann-Whitney [WRH+99] (car les données ne suivent pas une distribution normale). Le Tableau 13.6 présente les résultats de ce test. Les graphiques présentés en Figures 13.5a et 13.5b permettent d'estimer les différences existant entre les types de tâches. La *p-value* est extrêmement faible et largement inférieure à 0.01 (4.472e-16). Il est donc possible de rejeter l'hypothèse H_0a avec une probabilité d'erreur inférieure à 0.01 et donc d'affirmer H_{1a} :

Conclusion 14. ***La Distance Relative entre Chemins Visuels est, en moyenne, plus faible pour les tâches de modification que pour les tâches de compréhension.***

Néanmoins, il est important de remarquer l'existence de valeurs extrêmes (visibles sur la Figure 13.5b), valeurs qui influencent donc fortement la moyenne.

13.6 Conclusions

L'analyse des métriques relatives aux données fournies par l'oculomètre nous permettent de tirer plusieurs conclusions. Toutes les conclusions concernant les mesures basées sur les fixations (Conclusions 8 et 11) nous permettent de nous assurer que les données récoltées durant l'expérience se comportent bien comme espéré. En effet, le *framework* influence fortement l'aspect visuel du diagramme, il est donc évident que celui-ci influence de manière significative les données oculométriques.

La non-influence du patron architectural sur la densité spatiale (Conclusion 11) et les *p-values* de l'influence du patron architectural sur les données oculométriques (Tableau 13.1) ne nous apportent que peu d'information et ne nous permettent pas de conclure sur une quelconque influence notable du patron architectural.

L'influence du sexe sur la durée moyenne des fixations (Tableau 13.2, bien qu'étant significative, n'est pas retenue comme pertinente et cela dû aux faiblesses de l'oculomètre (comme détaillé en Section 13.1). Aucune autre influence du sexe sur les données oculométriques n'est ressortie de notre analyse.

Une influence du niveau d'étude est notable. En effet, le niveau d'étude semble influencer de manière significative ($\alpha < 0.07$) quatre des cinq métriques considérées. Néanmoins, seules deux variables ont un seuil inférieur à $\alpha = 0.05$. Ces deux variables, que sont la densité spatiale et l'enveloppe convexe, s'accordent sur le même fait : les Masters ont une densité spatiale et une enveloppe convexe significativement plus faible que les Doctorants (Conclusions 9 et 12). Notons que les Masters ont également une densité de transition et une durée moyenne de fixation plus faible. Tous ces éléments nous permettent de supposer que **les Masters se concentrent plus sur des points spécifiques du diagramme**. Cependant, étant donné que la métrique IN AORI / IN AOII normalisé n'a fourni aucun résultat significatif, il n'est pas possible de s'assurer que ces points sont des points pertinents.

Quant à la métrique que nous avons proposée en Section 9, le niveau d'étude et le type de tâche influencent significativement la distance entre les chemins visuels (Tableau 13.3 et Conclusions 13 et 14). Pour le niveau d'étude, **les Masters semblent se comporter d'une façon plus homogène que les Doctorants**. Cette conclusion concorde avec l'analyse effectuée sur les données collectées par Van den Plas [Van09] et présentée en Section 9.3. Enfin, pour le type de tâche, le comportement des sujets du point de vue du parcours du diagramme est significativement plus homogène pour les tâches de modification que pour les tâches de compréhension.

CHAPITRE 14

Analyse des mesures subjectives

LE dernier aspect de notre analyse concerne l'approche subjective de l'évaluation de la charge mentale ressentie par les sujets lors des tâches qu'ils ont effectué durant notre expérience. La mesure subjective des sujets que nous utilisons est NASA-TLX (voir Section 5.3).

Ce chapitre détaille les résultats que nous avons obtenus à l'aide de cette méthode, l'analyse de ces données et les conclusions que nous tirons de ces résultats. L'analyse se présente sous deux grands axes : d'une part, l'analyse de l'influence de la variable indépendante "variante du patron architectural" sur la variable dépendante MW, et d'autre part, l'analyse de l'influence du sexe et du niveau d'étude sur cette variable dépendante.

Sommaire

Échelle	Poids
Mental demand	5
Physical demand	0
Temporal demand	3
Own performance	2
Effort	4
Frustration	1

TABLE 14.1 – Poids attribués à chaque sous-échelle de NASA-TLX

14.1 Approche et mise en place

Dans cette section, nous présentons la configuration de la métrique NASA-TLX que nous utilisons, telle que décrite dans le Chapitre 5. Comme introduits en Section 11.3.1, les sujets évaluent leur MW à l'aide d'un questionnaire à la fin de chaque série de questions portant sur un diagramme (*JFreeChart* et *Swing*). Il nous est donc possible de comparer la MW subjective de chaque sujet pour chaque couple (Framework,Patron) qui lui a été attribué (par exemple, le couple (Swing,MVP)).

Dans le but de pouvoir obtenir une métrique représentative de la MW subjective, le mode d'emploi de TLX [NAS03] impose de fixer des poids pour chacune des sous-échelles. Les poids doivent se trouver dans l'intervalle $[0..5]$, ceux-ci étant détaillés dans le Tableau 14.1 et justifiés ci-dessous (les sous-échelles sont détaillées en Section 5.3) :

Demande mentale : Dans notre expérimentation, la demande mentale possède le poids le plus élevé. En effet, nous estimons que cette échelle est la plus représentative de la charge mentale du sujet durant les tâches de maintenance.

Demande physique : Un poids nul est attribué à cette sous-échelle car le sujet est assis durant toute l'expérience et n'est pas entravé dans ses mouvements par l'oculomètre.

Demande temporelle : Le temps nécessaire pour effectuer une tâche de maintenance est une variable importante. Le poids de 3 lui est donc attribué afin de donner de l'importance à l'estimation de la demande temporelle par le sujet sans pour autant lui en donner trop. Le fait qu'aucune pression temporelle ne soit présente durant notre expérience justifie le choix d'un poids moyen pour cette sous-échelle.

Performance : L'évaluation de la performance par le sujet est un élément à prendre en compte dans l'évaluation de la MW subjective. Cependant, nous évaluons non seulement le temps mais également l'exactitude des réponses des sujets. Nous avons donc attribué un poids de 2 à cette sous-échelle afin de ne pas empiéter excessivement sur les autres mesures.

Effort : La sous-échelle d'effort est une évaluation globale de l'effort fourni afin d'effectuer les tâches demandées. Un poids élevé lui est donc attribué.

Frustration : La frustration pouvant être évaluée en fonction de la question et non pas réellement en fonction du diagramme, et le fait que cette mesure nous semble peu pertinente, justifie notre choix de lui attribuer un poids relativement faible.

Une fois ces poids posés, il est possible de calculer la MW à l'aide de la formule suivante (Section 5.3) :

$$\mathrm{MW} = \mathrm{MD} \times \mathrm{MDW} + \mathrm{PD} \times \mathrm{PDW} + \mathrm{TD} \times \mathrm{TDW} + \mathrm{OP} \times \mathrm{OPW} + \mathrm{FR} \times \mathrm{FRW} + \mathrm{EF} \times \mathrm{EFW}$$

Le Tableau A.5 détaille les résultats obtenus.

14.2 Étude de l'influence des variables indépendantes

Au regard de la Section 11.2, les deux variables indépendantes sont le *framework* et le patron architectural. Ces deux facteurs disposent respectivement de deux (Swing et JFreeChart) et trois (MVC,MVP et MD) traitements possibles.

14.2.1 Hypothèses

H_{0a}	Les tâches de maintenance nécessitent, en moyenne, le même niveau de MW subjective quelle que soit la variante X du patron architectural utilisée.
H_{0b}	Les tâches de maintenance nécessitent, en moyenne, le même niveau de MW subjective quelle que soit la variante Y du *framework* utilisé.
H_{0c}	Les tâches de maintenance nécessitent, en moyenne, le même niveau de MW subjective quelle que soit la variante X du patron architectural utilisée et quel que soit le *framework* Y utilisé.
H_{1a}	La variante X nécessite, en moyenne, un niveau de MW subjective significativement plus faible que les autres pour les tâches de maintenance.
H_{1b}	Le *framework* Y nécessite, en moyenne, un niveau de MW subjective significativement plus faible que les autres pour les tâches de maintenance.
H_{1c}	La variante X nécessite, en moyenne, un niveau de MW subjective significativement plus faible que les autres pour les tâches de maintenance quel que soit le *framework* Y utilisé.
	$X \in \{\text{MVC}, \text{MD}, \text{MVP}\}$
	$Y \in \{\text{JFreeChart}, \text{Swing}\}$

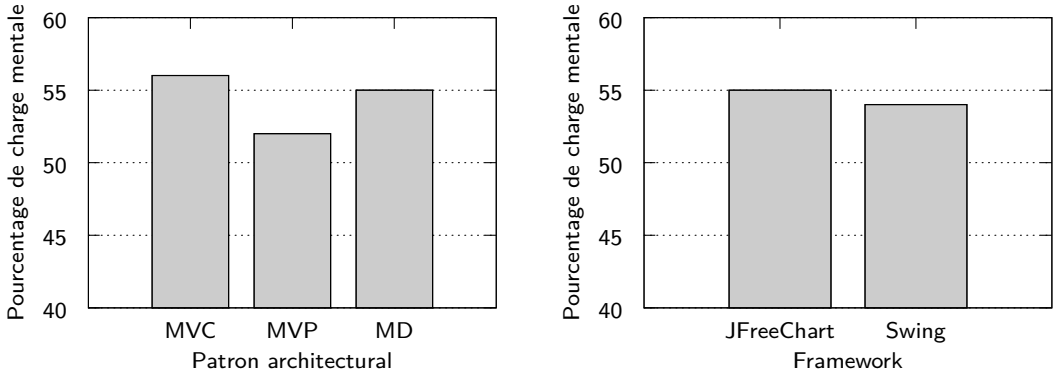
14.2.2 Données

Le Tableau A.5 présente la MW subjective calculée directement sur base des réponses fournies par les sujets. Le Graphique 14.1a est la représentation graphique de la moyenne des MW subjectives par variante de patron architectural. Nous pouvons observer que le patron architectural MVP a la MW subjective la plus faible. Les patrons MVC et MD ont une MW subjective fortement semblable. Néanmoins, les variations de la MW subjective sont relativement faibles. Ces données sont utilisées afin de tester l'hypothèse H_{0a}.

Dans le but de tester l'hypothèse H_{0b}, nous utilisons les données présentées par le Graphique 14.1b qui présente la moyenne des MW subjectives par *framework*.

14.2.3 Test d'hypothèse

Les tests d'hypothèses utilisés dans cette section sont tous ANOVA.



(a) Selon le patron architectural

(b) Selon le *framework*

FIGURE 14.1 – Moyenne des charges mentales subjectives

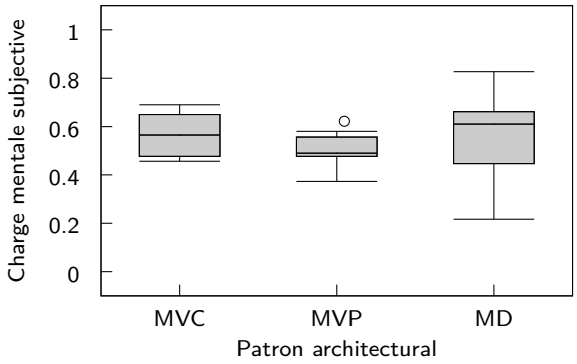


FIGURE 14.2 – Boîtes à moustaches comparant la dispersion de la MW subjective en fonction du patron architectural

	Patron	Residu
Degrés de liberté	2	27
Somme des carrés	0.01179	0.52065
Moyenne des carrés	0.0058942	0.0192831
F-value	0.3057	
Pr(>F)	0.7391	

TABLE 14.2 – Test d'hypothèse ANOVA pour la MW subjective en fonction du patron architectural

Test de l'hypothèse H_{0a}

Le Tableau 14.2 présente les résultats du test d'hypothèse. La Figure 14.2 permet de visualiser la dispersion des échantillons par patron architectural. La *p-value* est de 0.7391, il est donc impossible de rejeter H_{0a}.

Test de l'hypothèse H_{0b}

Le Tableau 14.3 présente les résultats du test d'hypothèse. La Figure 14.3 permet de visualiser la dispersion des échantillons par *framework*. La *p-value* est ici de 0.9686, ce qui rend impossible de rejeter H_{0b}. Néanmoins, il est possible de rejeter H_{1b}. Nous pouvons donc affirmer H_{0b} :

Conclusion 15. ***Les tâches de maintenance nécessitent en moyenne le même niveau de MW subjective quelle que soit la variante Y framework utilisée.***

Test de l'hypothèse H_{0c}

Le Tableau 14.4 résume les résultats du test d'hypothèse et également le test appliqué aux effets cumulés du patron architectural et du *framework*. La *p-value* vaut ici 0,1. Cette valeur est trop élevée que pour pouvoir rejeter l'hypothèse H_{0c}.

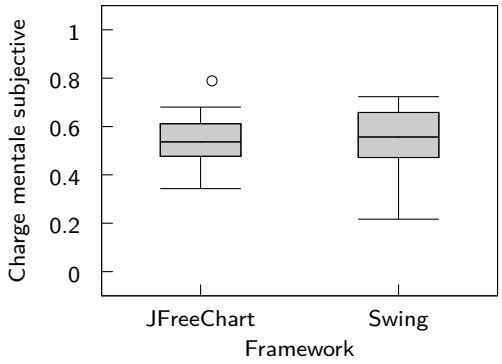


FIGURE 14.3 – Boîte à moustaches comparant la dispersion de la MW subjective en fonction du *framework*

	Patron	Residu
Degrés de liberté	1	28
Somme des carrés	0.00003	0.53240
Moyenne des carrés	0.00003	0.019014
F-value	0.0016	
Pr(>F)	0.9686	

TABLE 14.3 – Test d'hypothèse ANOVA pour la MW subjective en fonction du *framework*

	Patron	Framework	Patron/Framework	Residu
Degrés de liberté	2	1	2	24
Somme des carrés	0.01179	0.00003	0.08844	0.43192
Moyenne des carrés	0.005894	0.00003	0.044219	0.017997
F-value	0.3275	0.0016	2.4570	
Pr(>F)	0.7239	0.9686	0.1070	

TABLE 14.4 – Test d'hypothèse ANOVA pour la MW subjective en fonction du *framework*, du patron architectural et cumulés

Ce test d'hypothèse conclut l'analyse des variables indépendantes en affirmant que le *framework* n'influence pas la MW subjective des sujets.

14.3 Étude de l'influence des facteurs confondants

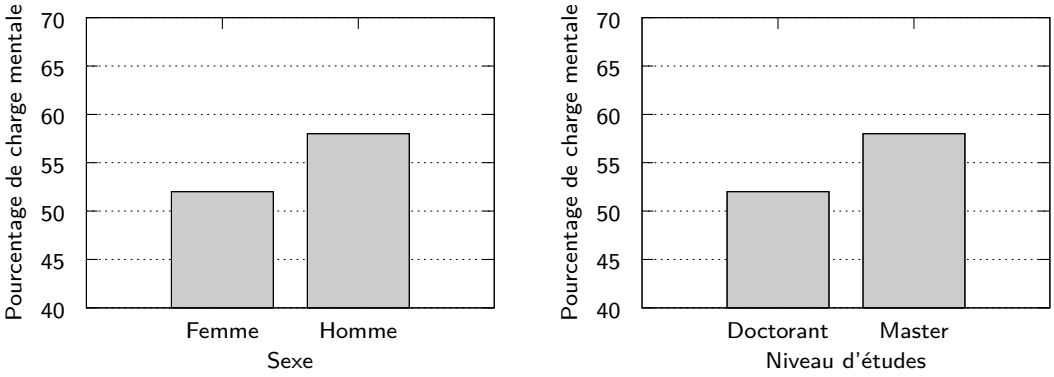
Dans cette section, nous présentons les résultats de l'analyse de l'influence du sexe et du niveau d'étude sur la MW subjective.

14.3.1 Hypothèses

H_{0a}	Les tâches de maintenance nécessitent, en moyenne, le même niveau de MW subjective quel que soit le sexe du sujet.
H_{0b}	Les tâches de maintenance nécessitent, en moyenne, le même niveau de MW subjective quel que soit le niveau d'étude du sujet.
H_{1a}	Le niveau de MW subjective significativement plus faible, en moyenne, pour un sexe que pour l'autre.
H_{1b}	Le niveau de MW subjective significativement plus faible, en moyenne, pour un certain niveau d'étude.

14.3.2 Données

Afin de tester l'hypothèse H_{0a}, les données utilisées sont celles présentées par le Graphique 14.4a et celles du Graphique 14.4b pour l'hypothèse H_{0b}.



(a) Selon le sexe

(b) Selon le niveau d'étude

FIGURE 14.4 – Comparaison de la MW subjective selon les facteurs confondants

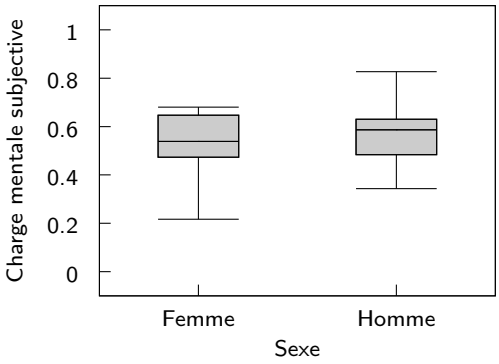


FIGURE 14.5 – Boîte à moustaches comparant la dispersion de la MW subjective en fonction du sexe du sujet

W	P-value
90.5	0.3825

TABLE 14.5 – Test d'hypothèse Mann-Whitney pour l'influence du sexe sur la MW subjective

14.3.3 Test d'hypothèse

Le test d'hypothèse effectué ici est Mann-Whitney.

Test de l'hypothèse H_{0a}

Le Tableau 14.5 présente les résultats du test d'hypothèse H_{0a}. Le Graphique 14.5 illustre la répartition de la MW subjective en fonction du sexe du sujet. La valeur de la *p-value* est ici de 0.3825, valeur trop élevée que pour rejeter l'hypothèse H_{0a}.

Test de l'hypothèse H_{0b}

Le Tableau 14.6 présente les résultats du test d'hypothèse. Le Graphique 14.6 illustre la répartition de la MW subjective en fonction du niveau d'étude du sujet. La valeur de la *p-value* (0,33) ne permet pas de rejeter l'hypothèse H_{0b}.

14.4 Conclusions

Les différents tests d'hypothèses effectués dans cette section nous permettent d'affirmer que le *framework* utilisé durant nos expériences n'a aucune influence significative sur la charge mentale des sujets. L'étude des facteurs confondants que sont le sexe et le niveau d'étude des sujets n'ont permis de tirer aucune conclusion.

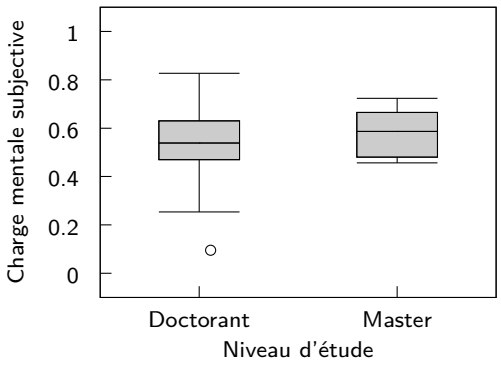


FIGURE 14.6 – Boîte à moustaches comparant la dispersion de la MW subjective en fonction du niveau d'étude

W	P-value
84.5	0.33

TABLE 14.6 – Test d'hypothèse Mann-Whitney pour l'influence du niveau d'étude sur la MW subjective

CHAPITRE 15

Rétrospective de l'expérience

> Les savants sont des êtres humains. C'est une vérité qu'on oublie souvent, mais qu'on peut vérifier par l'expérience.
>
> George Mac Bundy

UNE fois l'expérience effectuée et l'analyse des résultats réalisée, il est nécessaire d'effectuer une synthèse des trois types de mesures effectuées (mesure de performance de la tâche, mesures physiologiques et mesures subjectives), c'est le but de la section 15.1. Il est également primordial d'énoncer les menaces à leur validité. Il s'agit du rôle de la Section 15.2. Ensuite, nous rapportons dans la Section 15.3 des leçons diverses que nous avons tirées de la réalisation et de l'exécution de l'expérience.

Sommaire

15.1 Conclusions générales

L'analyse des données sous les trois aspects différents que sont la performance de la tâche principale (Section 12), les mesures physiologiques (Section 13) et les mesures subjectives (Section 14), nous permettent de conclure les éléments suivants.

Premièrement, **le *framework* (Swing ou JFreeChart) a une influence significative sur le temps de réponse**. En effet, les tâches de maintenance effectuées sur le *framework* JFreeChart ont été réalisées, en moyenne, significativement plus rapidement que celles effectuées sur Swing. Les données oculométriques confirment également que le *framework* a une influence significative sur toutes les métriques étudiées basées sur les fixations. La seule analyse n'ayant pas donnée de résultat significatif au sujet du *framework* est l'évaluation de la charge mentale subjective. Ces résultats semblent cohérents car à chaque *framework* correspond un diagramme différent. Néanmoins, ces analyses ne sont pas utiles si ce n'est pour nous conforter dans le choix de nos métriques, dans la validité de la collecte des données et dans la construction des questions et des diagrammes. En effet, le fait que les mesures effectuées respectent bien des prédictions triviales est encourageant pour la suite des analyses. La complexité [1] des diagrammes de Swing étant plus élevée que celle de JFreeChart (au regard de la Section 10.5.6) implique, par exemple, que les sujets nécessitent plus de temps pour réaliser les tâches sur Swing.

Deuxièmement, **aucune influence du patron architectural** n'est apparue dans les résultats, qu'il s'agisse du temps de réponse, de la validité des réponses, des métriques basées sur les fixations ou de la charge mentale subjective.

Troisièmement, la **non-influence du sexe** est mise en exergue par les résultats. Effectivement, le sexe n'a pas d'influence significative ni sur le temps de réponse, ni sur les métriques oculométriques.

Quatrièmement, les résultats montrent que **le niveau d'étude (Master ou Doctorant) influence significativement le temps de réponse, plusieurs variables oculométriques et la distance relative entre les chemins visuels**. Ce résultat est sans doute l'apport le plus important de notre part en ce qui concerne la récolte et l'analyse des données. Nous mettons en avant le fait que **les Masters prennent moins de temps pour répondre, que les zones qu'ils observent sont plus réduites et plus ciblées et qu'ils se comportent d'une manière plus homogène que les Doctorants**. Ces conclusions sont confirmées par l'application de notre métrique aux données récoltées par Van den Plas [Van09] : les débutants se comportent également d'une façon plus homogène que les experts lorsqu'il s'agit de parcourir un diagramme UML afin de réaliser des tâches de maintenance. Nos résultats confortent également les observations faites par Yusuf et al. [YKM07]. En effet, ceux-ci avaient observé durant leurs expérimentations que les sujets semblaient se comporter différemment en fonction de leur expérience. Cependant, étant donné que la métrique IN AORI / IN AOII normalisé (métrique permettant d'estimer la pertinence de l'effort visuel) n'est pas influencée par le niveau d'étude et qu'il n'existe pas de différence significative dans le taux de bonnes réponses entre Masters et Doctorants, il n'est pas possible d'estimer si la "plus grande précision" des Masters est pertinente ou non.

1. En terme de nombre de classes, par exemple.

15.2 Menaces à la validité

Cette section a pour but de présenter les différentes menaces à la validité de notre expérience que nous avons pu identifier. Ces menaces permettent de remettre en contexte l'expérience réalisée et de mettre en avant les limites de celle-ci. Cette section se décompose en quatre parties (suivant la méthode présentée dans la Section 6.3.3 de [WRH+99]) : la **validité interne** vérifie que le traitement implique les résultats, la **validité externe** s'assure que le résultat de l'étude peut être généralisé hors de la portée de l'étude, la **validité de construction** s'intéresse au lien entre entre la théorie et les observations et la **validité de conclusion** concerne la relation entre le traitement et les résultats.

15.2.1 Validité interne

La **courbe d'apprentissage** est un élément à ne pas négliger lorsqu'il s'agit d'analyser les résultats. Celle-ci peut en effet influencer grandement les données collectées. Dans notre cas, l'effet d'apprentissage peut apparaître au fur et à mesure de la réalisation des tâches sur un même diagramme. L'effet d'apprentissage étant relativement réduit lorsque le sujet passe au diagramme suivant. Nous avons choisi de commencer le questionnaire par des questions de compréhension, afin que le sujet puisse prendre connaissance du diagramme avant de le modifier. De ce fait, nous contrôlons la courbe d'apprentissage autant que possible afin de la rendre **équivalente pour tous les sujets**.

Cependant, il est tout à fait possible que certains sujets aient déjà travaillé avec Swing ou JFreeChart. Le fait de connaître le fonctionnement d'un logiciel avant d'avoir pu prendre connaissance des diagrammes UML peut grandement biaiser les résultats de l'expérience. Il ne serait pas impossible, par exemple, qu'un sujet connaisse la réponse à une question de compréhension avant même le début de l'enregistrement par l'oculomètre. Ce biais influencerait donc toutes les mesures effectuées durant l'expérience, qu'elles soient de performance, oculométriques ou subjectives. Afin d'éviter cette influence, nous demandons, avant chaque début d'expérience, si le sujet connaît les *frameworks* sur lesquels portent les tâches de maintenance. Alors que Swing est utilisé par la plupart des sujets au cours de leurs développements Java, JFreeChart est moins connu. Tel qu'expliqué Section 10.1.1 (cf. la contrainte **C6** de cette section), les logiciels ou les *frameworks* soumis à l'étude peuvent être *utilisés* par les sujets mais ne doivent pas être *étudiés* par ceux-ci. Swing et JFreeChart respectent cette contrainte et laissent donc supposer que les sujets n'en ont pas une connaissance assez approfondie pour pouvoir répondre instinctivement aux tâches de maintenance. Effectivement, les tâches demandées sur ces deux *frameworks* sont rarement effectuées par de simples utilisateurs. De par ces choix, cette menace est amoindrie.

La fatigue ressentie par un sujet durant l'expérience et la diminution de la concentration en fonction du temps peuvent être vues comme des menaces à la validité interne de notre expérience (ces effets sont appelés de **maturation** par [WRH+99]). En effet, l'expérience durant en moyenne une heure, certains sujets ont mentionné un état de fatigue à la fin de celle-ci. La fatigue est un élément qui peut perturber les mesures oculométriques. Effectivement, nous avons remarqué durant l'expérience que certains sujets avaient tendance à légèrement fermer les yeux ou à les plisser afin de mieux voir le diagramme. Ces actions ont pour conséquence de perturber l'oculomètre qui peut ignorer certaines fixations. Dans le but de limiter les effets de la fatigue, une pause était faite entre les deux types de diagrammes afin que le sujet ait la possibilité de reposer ses yeux lorsqu'il répond au questionnaire NASA-TLX. Nous avons également limité la durée de l'expérience à 1h30 sans en avertir le sujet (afin de ne pas introduire de notion de pression temporelle) dans le cas où un sujet prendrait trop de

temps. Aucun sujet n'a dépassé ce temps limite. Cette menace est, par conséquent, limitée.

15.2.2 Validité externe

Différentes menaces peuvent venir entraver la généralisation des résultats obtenus durant cette expérience. Celles-ci sont nommées **Interaction de la sélection et du traitement** par [WRH+99]. Un des éléments majeurs est l'homogénéité des sujets. Tels que décrits dans la Section 10.4, ils proviennent tous d'un milieu académique et ne sont donc pas représentatifs de l'industrie du logiciel. Bien que l'homogénéité des sujets diminue la validité externe, celle-ci augmente la validité de conclusion.

Le fait que l'expérience ne repose que sur deux types de *framework* est également un élément qui empêche une généralisation des résultats. Il serait nécessaire d'appliquer notre expérience à des projets industriels réels afin de pouvoir tirer des conclusions plus générales. Cependant, ce type d'expérience est extrêmement coûteuse et souvent irréalisable [WRH+99, p. 48].

15.2.3 Validité de construction

En tant qu'expérimentateurs, nous avons conçu les diagrammes et les questions. Cela peut entraîner un biais involontaire dans les conclusions tirées au sujet de l'influence des patrons architecturaux sur la maintenabilité. En effet, il est tout à fait possible que nous ayons créé inconsciemment des questions et des tâches de maintenance plus faciles à réaliser sur un certain type de patron architectural étant donné que nous connaissions les avantages et inconvénients réputés de chaque type. Par exemple, le patron architectural MVC concentre la majorité de la gestion des évènements au sein de son composant Contrôleur. Or, certaines questions portent directement sur la modification de ces évènements. Le patron MVP employé dans nos diagrammes permet la présentation des données à l'interface graphique. Or, certaines questions se focalisent sur l'affichage des données sans leur modification. Pour réduire notre influence sur les questions, elles ont préalablement été révisées et supervisées par des personnes ne participant pas à l'expérience.

Pour réduire le biais provoqué par la menace des diagrammes, ceux-ci sont initialement obtenus par une méthode de rétro-ingénierie telle que décrite en Section 10.5.1. Cette méthode diminue notre influence sur les diagrammes. De plus, l'injection des patrons architecturaux dans les différents diagrammes suit une méthodologie prédéfinie et majoritairement toujours identique dans le but de diminuer un ascendant subjectif de notre part. Ces menaces tombent dans la catégorie **attentes des expérimentateurs**[WRH+99].

Les menaces sociales ne sont pas à négliger car elles peuvent influencer de manière significative les résultats et les conclusions. La majorité des sujets de notre expérience se connaissent entre eux étant donné qu'ils font partie du même laboratoire de recherche. De ce fait, il est possible que des sujets aient commenté notre expérience à de futurs sujets. Ces commentaires peuvent avoir une influence sur le stress ressenti par les sujets avant le début de l'expérience. Le cas où les sujets ayant déjà passé l'expérience précisent aux futurs sujets que le test est relativement difficile en est un exemple.

L'**appréhension de l'évaluation** a été remarquée durant l'exécution de l'expérience. Nous avons en effet observé que quelques sujets semblaient inquiets quant à leurs résultats et à leur diffusion. Par conséquent, il est raisonnable de penser que certains sujets ont été tentés de minimiser leur

charge mentale par exemple, pour avoir de “meilleurs" résultats. Le fait que le sujet sache que ses mouvements oculaires sont mesurés peut également avoir une influence sur son comportement. Il peut tenter, inconsciemment, de ne regarder qu'aux bons endroits, réduisant ainsi les fixations dans les zones non pertinentes. Par exemple, un sujet n'analysera pas une classe du diagramme par curiosité *durant l'expérience* s'il sait qu'elle ne correspond pas à la tâche demandée. Toutefois, peu de sujets ont présenté cette inquiétude.

Malgré le fait que les données soient anonymes, des sujets pourraient **tricher** durant l'expérience afin d'améliorer leurs résultats. Une des façons les plus simples de biaiser les résultats pour un sujet est de valider le fait d'avoir trouvé une réponse alors qu'il ne l'a pas trouvée. Pour ce faire, il suffit au sujet d'appuyer sur la touche `ENTER` (comme décrit en Section 11.3.1) afin d'arrêter l'enregistrement et de continuer à chercher la réponse sur le diagramme devant servir uniquement de support pour l'écriture de la réponse. Pour empêcher ce type de comportement, l'enregistrement des coordonnées visuelles continue durant la phase de réponse [2]. Cela nous permet de vérifier si le sujet a continué à chercher la réponse (fixations éparses sur le diagramme) ou si celui-ci a réellement trouvé la réponse (fixations regroupées sur le ou les classes contenant la réponse). Aucun sujet n'a triché de cette façon, cette menace est donc écartée.

Palmer [Pal99, p. 532] précise dans son ouvrage qu'il est possible de porter son attention sur un point qui est hors du point (cercle) de vision (la fovéa). En théorie, il serait donc possible qu'un sujet porte son attention en un endroit alors que son regard se porte sur un autre. Cependant, effectuer ce type d'opération est relativement difficile et est totalement contre-intuitif. Nous supposons qu'aucun sujet n'a tenté de tromper les résultats de l'oculomètre en fixant un endroit différent de l'endroit de l'attention.

L'**effet Hawthorne** (ou expérience Hawthorne) est une situation dans laquelle les résultats tirés d'une expérience ne sont pas dûs aux facteurs expérimentaux (variables) mais bien au simple fait que le sujet, étant conscient qu'il est observé, est plus motivé et se comporte différemment que s'il était seul. Cet effet est observé dans de nombreuses expériences et peut pousser à une augmentation de la productivité. Toutefois, dans notre cas, la plupart des sujets sont habitués à ce genre d'expérience car il s'agit d'une spécialité du laboratoire. De plus, l'environnement utilisé est connu par la majorité des sujets vu qu'il s'agit de leur lieu de travail. En outre, une discussion préliminaire à l'expérience permet de mettre les sujets à l'aise et de réduire la pression qu'ils pourraient ressentir. Enfin, même s'ils sont couramment surveillés durant l'expérience, l'agencement du laboratoire (Section 10.6.3) permet au sujet de se sentir à l'aise, seul face à ses tâches de maintenance, mais pouvant à tout moment obtenir notre aide s'il a des questions. La présence des expérimentateurs est ainsi réduite au minimum du point de vue du sujet.

Les tâches de maintenance ne sont pas effectuées sur un code source mais sur des diagrammes de classes UML. Ce choix peut sembler être une menace car effectuer une tâche sur un diagramme n'est évidemment pas équivalent à l'effectuer sur un code source écrit dans un ou plusieurs langages de programmation. Néanmoins, les diagrammes UML s'imposent depuis plusieurs années comme un standard de modélisation pour les systèmes orientés objet [Eic03, Gué06]. En outre, le rejet des sujets n'ayant pas des connaissances suffisantes en UML nous assure que ce formalisme n'est pas un frein

2. Pour rappel, le diagramme est toujours affiché lorsque le sujet écrit sa réponse (une fois la touche `ENTER` enfoncée). Le diagramme a alors la vocation de supporter le sujet si celui-ci a oublié un nom de classe pour sa réponse, par exemple. Bien entendu, ces données ne sont pas utilisées durant l'analyse statistique.

à l'exécution des tâches.

15.2.4 Validité de conclusion

Durant les tests d'hypothèses réalisés dans les Sections 12, 13 et 14, nous avons décidé de conserver la totalité des données qui étaient analysables. Nous entendons par là que nous n'avons pas exclu les *outliers* de l'analyse. Ceux-ci sont visibles sur les différentes boîtes à moustaches sous forme de cercles ∘. La moyenne étant fort sensible aux extrêmes, les *outliers* peuvent avoir un effet non négligeable sur celle-ci. Un nombre plus important d'*outliers* est présent dans l'analyse de la distance entre chemins visuels (Figures 13.4b et 13.5b). Plusieurs théories s'affrontent afin de savoir s'il est nécessaire de supprimer les *outliers* lors de l'analyse (voir la Section 6.3.5). Nous avons décidé de les conserver car aucune cause n'a été identifiée pour justifier ces valeurs extrêmes. Néanmoins, cette décision peut diminuer la **puissance statistique** des tests.

Une autre menace à la validité de conclusion est la **fiabilité des mesures**. Dans notre cas, un des problèmes majeurs que nous ayons rencontré (comme détaillé en Section 11.3.2) est lié à l'utilisation de l'oculomètre. Certains sujets portent des lunettes et cela est problématique et peut biaiser l'enregistrement de leurs fixations et de leurs saccades. Afin d'éliminer cette menace à la validité, nous avons corrigé manuellement les fixations des sujets posant problème comme précisé en Section 11.3.2. De plus, les sujets avec lesquels ce problème était trop important ont été supprimés de l'ensemble des sujets pour l'analyse des mesures physiologiques. Cette menace est donc écartée.

En outre, les clignements des yeux des sujets peuvent également influencer les fixations de manière anecdotique. Les fixations manquantes durant les clignements des yeux d'un sujet ne sont pas à prendre en compte étant donné que le sujet ne visualise pas le diagramme durant ces microsecondes. Il ne s'agit donc pas d'une menace.

Le choix des termes utilisés pour formuler les tâches à effectuer peut être vu comme une menace à la **fiabilité des mesures**. En effet, si le sujet ne comprend pas correctement la question, des biais peuvent apparaître. Afin de minimiser cette menace, le questionnaire était pourvu d'une introduction posant le vocabulaire utilisé dans les tâches. De plus, le questionnaire était disponible en plusieurs langues (voir Section 10.7.2) et nous étions présents durant toute la durée de l'expérience afin de répondre aux éventuelles questions des sujets. En outre, la relecture et la validation des questions par M. Guéhéneuc, habitué à de telles démarches expérimentales, diminue cette menace à la validité.

Le *plantage* du logiciel récoltant les données oculométriques est potentiellement une menace à la validité, en effet, quelques données concernant quelques sujets ont été perdues à cause de ce problème.

L'**hétérogénéité des sujets** est un facteur présent dans toute expérience. Si le groupe est fortement hétérogène, les observations effectuées peuvent venir des différences existant entre les sujets et non des variables indépendantes. Dans notre cas, nous nous sommes assurés d'avoir un ensemble de sujets relativement uniforme, ce qui écarte cette menace. Cependant, cette homogénéité des sujets a un effet sur la validité externe étant donné qu'il est plus difficile de pouvoir généraliser les observations effectuées.

15.3 Leçons apprises

Cette section a pour but de présenter les leçons que nous avons apprises durant la réalisation et l'exécution de l'expérience.

L'oculométrie est un domaine en évolution, les améliorations apportées à chaque nouvelle génération d'appareils permettent une meilleure précision, une meilleure fiabilité et une calibration plus simple tout en réduisant les contraintes sur le sujet. L'oculométrie est une science à fort potentiel comme le montrent les récentes innovations dans le domaine de la visualisation 3D. Effectivement, la firme sud-coréenne LG a dévoilé, en juillet 2011, le premier moniteur combinant une technologie de 3D sans lunettes à une caméra exploitant le principe d'oculométrie. Son objectif est d'optimiser l'expérience en relief de l'utilisateur [LG11]. Cependant, l'oculométrie reste une technologie relativement jeune et sa fiabilité laisse encore à désirer. Son utilisation peut s'avérer extrêmement précieuse, mais les données récoltées doivent être analysées avec précaution. Des erreurs et des manques de précision peuvent apparaître et doivent être corrigés avant toute analyse. Baser entièrement une étude sur des mesures oculométriques nous semble quelque peu risqué. Ces mesures doivent être complétées par d'autres mesures (subjectives, par exemple) afin de donner plus de poids aux conclusions tirées. Approfondir l'analyse subjective par une série d'entrevues aurait probablement été enrichissant.

Les systèmes d'oculométrie (Section 8.1) que nous avons eu l'occasion de manipuler sont des appareils relativement coûteux et peu répandus. De ce fait, il est très difficile de trouver de la documentation concernant leur utilisation sur le Web. Cette particularité nous fait réaliser l'importance de la documentation pour le transfert de la connaissance au sein d'une équipe. L'oculomètre FaceLAB (Section 8.1.2) ayant été acquis par le laboratoire *Ptidej* seulement quelques mois avant notre arrivée sur place. Celui-ci n'avait été manipulé que par une seule personne, possédant de ce fait tout le savoir concernant l'utilisation et la configuration de l'appareil. Cette personne ne faisant plus partie du laboratoire à notre arrivée et n'ayant pas créé de documentation à destination des futurs utilisateurs, nous avons été contraints de contacter la personne-ressource (responsable de la vente de l'appareil) afin d'obtenir la démarche de configuration de l'appareil. Pour que cette situation ne se reproduise pas, nous avons créé une suite complète de documentation (*screencasts*, manuel de configuration, vidéos d'exemples [3]) à destination des futurs utilisateurs de l'oculomètre.

Lors de l'implémentation de TAUPE, l'écriture de tests unitaires s'est révélé indispensable. Ils nous aident non seulement à assurer une meilleure maintenabilité du programme (il est très simple de vérifier qu'aucun effet de bord n'est apparu lors de l'implémentation d'une nouvelle fonctionnalité), mais ils nous permettent également de détecter des erreurs de calculs qui auraient totalement modifié la nature des résultats.

Une autre leçon que nous retirons de notre expérience est le manque de constance dans les connaissances des sujets, et ce malgré l'équivalence de leur diplôme. Nous avons sans doute eu tendance à sur-évaluer les compétences des sujets lors de la création des questions et tâches à effectuer sur les diagrammes. En effet, certains sujets (possédant pourtant un diplôme de Maître en sciences informatiques) nous ont avoué avoir eu de grandes difficultés à répondre aux questions [4]. Toutefois,

3. Disponibles à l'adresse `http://www.ptidej.net/research/taupe/videos/`

4. Une personne en particulier, éliminée pour ses faibles connaissances en UML, s'est d'ailleurs assurée à plusieurs reprises que ses résultats resteraient anonymes et ne seraient pas communiqués au responsable du laboratoire.

ces personnes restaient tout de même minoritaires.

Une des leçons les plus importantes que nous ayons retenue de notre expérience est la difficulté de mener à bien une telle démarche. Les différents choix effectués, les idées nécessaires à sa bonne réalisation sont des éléments critiques devant être justifiés, pensés et réfléchis avant d'être mis en place. Ces choix influencent de manière significative les résultats et donc les conclusions qui en sont déduites. De mauvais choix ou des simplifications trop importantes peuvent réduire de manière dramatique la portée des conclusions. L'encadrement de M. Guéhéneuc nous a permis de prendre du temps pour nous concentrer sur chaque élément constitutif de l'expérience.

Une première version des fichiers générés par TAUPE ne correspondait pas tout à fait à la méthode d'interprétation des données que nous utilisons. En effet, les données générées par TAUPE doivent uniquement être les variables dépendantes, mais ne doivent pas s'occuper de l'analyse. Par exemple, il n'est pas utile de générer des moyennes à partir du logiciel car cet aspect est géré par un logiciel réservé à cet effet. Le fait d'avoir de telles données dans les fichiers de sortie de TAUPE complexifie le logiciel et l'analyse des données lors de l'utilisation d'un outil tel que **R**. Ces erreurs nous ont d'ailleurs permis de réaliser qu'il est particulièrement difficile de prévoir à l'avance l'utilisation qui sera faite des données et donc l'importance d'un code facilement modifiable.

CHAPITRE 16

Conclusions et Travaux futurs

> Une conclusion, c'est quand vous en avez assez de penser.
>
> Herbert Albert Fisher

16.1 Réponses aux questions de recherche

Les six questions de recherche introduites au début de ce document ont pour vocation de structurer mais également de conclure notre travail en y apportant une réponse. Ces réponses sont décrites en détail tout au long de ce mémoire et sont résumées dans cette section.

Comment définir le terme de "maintenabilité" ? (Q1) À l'aide d'un chapitre qui lui est dédié (Chapitre 3), nous tentons de définir la maintenabilité dans le cadre des modèles de qualité logicielle. Néanmoins, si ce chapitre définit formellement le terme de "maintenabilité", nous ne l'utilisons pas intégralement lors de notre étude. Cette étude empirique menée dans le but d'évaluer l'impact sur la maintenabilité de patrons architecturaux se focalise sur la maintenabilité interne des produits selon les termes du modèle ISO/IEC 9126 [ISO99]. Notre étude tente donc d'évaluer le niveau d'effort requis pour modifier des programmes. En l'occurrence, les tâches de maintenance à réaliser sont de plusieurs types qui font référence à la décomposition en métrique de la maintenabilité selon ISO/IEC 9126 : l'*analysabilité* et la *changeabilité*.

Quelles théories et outils le génie logiciel offre t-il pour mesurer la maintenabilité d'un programme ? Ces approches sont-elles suffisantes ? D'autres domaines peuvent-ils apporter une aide à ces mesures ? (Q2) L'évaluation de l'effort, c'est-à-dire de la charge mentale, nécessaire à l'exécution de tâches de maintenance peut être effectuée de plusieurs façons. Ces méthodes sont résumées au Chapitre 5. Alors que la mesure de la performance est couramment employée, notre choix se porte non seulement sur cette mesure mais également sur l'analyse de mesures physiologiques et l'analyse de mesures subjectives.

Les mesures subjectives peuvent être évaluées à l'aide de NASA-TLX [NAS03, CCPE09] qui fournit un ensemble de questions et d'échelles utiles pour cette évaluation. Les mesures physiologiques, elles, sont moins courantes et nécessitent nombre de théories auxiliaires pour être obtenues et évaluées. Au sein des sciences cognitives, la théorie de la compréhension de programmes et la théorie de la vision, une fois unifiées, donnent lieu à la théorie de la Vision–Compréhension. Cette théorie, proposée par Guéhéneuc [Gué09], rend possible l'acquisition de données physiologiques et l'interprétation de celles-ci dans le cadre de la charge mentale.

Comment réaliser une approche empirique pour évaluer cette qualité ? Comment y inclure ces théories ? (Q3) Wohlin et al. [WRH+99] proposent une approche expérimentale pour le domaine du génie logiciel, elle est contextualisée et résumée au Chapitre 6. Ils décrivent la réalisation d'une expérience et l'interprétation des données qui y sont récoltées. Notre étude empirique exploite cette approche dans le but de réaliser une expérimentation dont l'objectif principal est l'évaluation de la charge mentale des sujets effectuant des tâches de maintenance. Cette charge mentale est évaluée selon les trois théories mentionnées ci-dessus, à savoir la *mesure de la performance*, la *mesure des données physiologiques* et la *mesure subjective*.

Lors des mesures de performance, nous utilisons le temps nécessaire à la réalisation d'une tâche ainsi que le taux de bonnes réponses des sujets. Les mesures physiologiques exploitent la théorie Vision–Compréhension et sont donc des mesures oculométriques. Ces métriques sont initialement la densité spatiale, la densité de transition, le *IN AORI/IN AOII* normalisé, la durée moyenne des fixations ainsi que l'enveloppe convexe des fixations. Enfin, NASA-TLX est utilisé pour l'acquisition des mesures subjectives des sujets.

Les outils disponibles sont-ils suffisants pour supporter ces théories ? (Q4) L'oculomètre FaceLAB décrit au Chapitre 8 utilisé lors de l'expérience permet d'obtenir des données pour deux types d'analyses. Effectivement, ce type de matériel permet d'obtenir les fixations et les saccades d'un sujet mais également la durée de réalisation d'une tâche. Un support papier rend possible l'acquisition des réponses des sujets (et donc des données d'exactitude des réponses aux tâches de maintenance) ainsi que des données subjectives pour NASA-TLX. L'analyse des données est facilitée par l'utilisation du logiciel **R**.

Si l'obtention des données oculométriques brutes est simple à l'aide de FaceLAB et de son outil GazeTracker™, le calcul des métriques oculométriques est fortement limité. Le développement du logiciel libre TAUPE par nos soins [DLG+11] y apporte une solution non seulement pour notre expérience mais également pour les prochaines recherches nécessitant des métriques relatives à l'oculométrie et la performance des tâches réalisées.

En terme de métriques oculométriques, l'introduction de notre nouvelle métrique *Distance Relative entre Chemins Visuels* répond à un manque en terme d'analyse dynamique de parcours visuels entre sujets.

Quelles alternatives et variantes existe t-il au patron Modèle-Vue-Contrôleur et comment sont-elles définies ? (Q5) Le Chapitre 2 présente les patrons de façon générale et plus précisément les patrons architecturaux Modèle-Vue-Contrôleur (MVC) [GHJV94], Modèle-Vue-Présentateur (MVP) [Pot96], *Model-Delegate* ou *Document-View* [BMR+96] (MD), *Presentation Model* [Fow04] et d'autres de façon moins détaillée. Néanmoins, les patrons considérés par notre étude sont les trois premiers : MVC, MVP et MD. Ce Chapitre apporte une terminologie claire au vu de la littérature disparate qui y fait référence.

Ces différents patrons architecturaux ont-ils un impact différent sur la maintenabilité ? (Q6) L'analyse de l'impact de ces trois patrons sur la maintenabilité interne des programmes est réalisée lors de l'étude empirique présentée aux Chapitres 10 et 11. Cependant, aucun résultat significatif n'est ressorti de ces analyses. Il n'est donc pas possible de prouver l'impact (ou le non-impact) de ces différents patrons architecturaux sur la maintenabilité. Au vu des analyses effectuées sur les patrons

architecturaux, le fait que certains résultats soient non-significatifs est majoritairement influencé par le faible nombre de sujets. Effectivement, concevoir notre expérience pour deux patrons (plutôt que trois) aurait potentiellement résolu ce problème.

Quelles caractéristiques des mainteneurs peut influencer cet impact ? (Q7) Les différentes analyses que nous avons effectuées (Chaptires 12, 13 et 14) tendent à mettre en évidence que le sexe d'un mainteneur n'a aucune influence sur ses performances lors de la réalisation de tâches de maintenance. Cependant, les résultats montrent que le niveau d'étude (Master ou Doctorat) influence significativement le temps de réponse, plusieurs variables oculométriques et la distance relative entre les chemins visuels. Ces données ont été confirmées par l'application de la métrique *Distance Relative entre les Chemins Visuels* sur les données de Van den Plas [Van09]. Les résultats obtenus confirment également les observations faites par Yusuf et al. [YKM07].

16.2 Travaux futurs

Certaines contraintes, qu'elles soient temporelles ou matérielles, nous ont empêché d'investiguer différents aspects que nous présentons ci-dessous. Ceux-ci pourraient donner lieu à de futures expériences et analyses.

L'oculomètre FaceLAB présenté en Section 8.1.2 permet d'enregistrer la dilatation de la pupille. Ces données peuvent être utilisées pour mesurer, par exemple, le niveau de fatigue d'un sujet [Pal99]. Durant notre analyse statistique des mesures oculométriques, nous n'avons pas utilisé ces données, mais il pourrait être intéressant de les analyser afin d'estimer le niveau de fatigue des sujets et donc de réduire la menace à la validité interne qu'est la fatigue des sujets.

Durant l'analyse des données, nous avons tenté de mettre en avant l'influence de certaines caractéristiques des sujets (telles que le sexe ou le niveau d'étude) sur les variables indépendantes. Il pourrait être intéressant de tenter d'organiser les individus en *clusters* en fonction des variables indépendantes pour ensuite vérifier si ces *clusters* correspondent à des caractéristiques des individus. Cette méthode serait particulièrement intéressante pour la métrique *Distance Relative entre Chemins Visuels*.

Concernant la métrique *distance relative entre chemins visuels*, l'utilisation d'une autre métrique (différente de la distance de Levenshtein) pour comparer deux chemins visuels peut être envisagée. Par exemple, la *Computing Similarity* [Gus97, p. 226] ou la *plus longue sous-séquence commune* (LCS) [CS75] permettrait de considérer des séquences de zones différemment que par l'algorithme de Levenshtein.

Afin de confirmer les différentes conclusions tirées lors de l'analyse des données, il serait utile de répliquer l'expérience avec d'autres individus et d'autres diagrammes, voire même d'autres variantes de MVC. Un nombre de sujet plus élevé pourrait permettre de donner plus de poids aux conclusions et d'accentuer leur caractère significatif (ou non). L'idée de comparer des diagrammes avec ou sans patron du point de vue de la maintenabilité peut également être envisagée. En outre, un plus grand nombre de sujets permettrait de tester une analyse sans *outliers*.

Tel que le précisent Wohlin et al. dans [WRH+99, p. 48], il est préférable d'analyser l'objet d'étude dans son environnement afin d'assurer des résultats généraux les plus valides possibles. En l'occurrence, il serait par conséquent intéressant d'étudier l'influence des variantes de MVC sur la

maintenabilité dans un milieu professionnel. Effectivement, même si les objets d'étude utilisés lors de notre expérience sont des logiciels largement utilisés tant par le milieu académique que par l'industrie, la totalité des sujets proviennent du domaine académique.

Différents algorithmes de différenciation entre saccades et fixations existent [SG00], il pourrait être intéressant de les appliquer aux données récoltées plutôt que de déterminer les fixations en fonction de leur durée. De plus, il serait potentiellement utile d'appliquer des algorithmes de détection de zones d'intérêt (tels que ceux présentés Privitera et Stark [PS00]) afin de vérifier si les zones attribuées par nos soins sont considérées comme pertinentes au niveau oculométrique.

16.3 Contributions

Bien que notre travail repose en partie sur une étude expérimentale dans le domaine du génie logiciel, il ne s'y résume pas. Cette section a pour objectif d'énoncer les contributions de notre travail dans différents domaines et de donner les références où elles sont décrites en détails.

16.3.1 Résultats de l'expérience

Le cœur de notre travail repose sur l'étude empirique réalisée au sein du laboratoire *Ptidej/Soccer* de l'École Polytechnique de Montréal. Les données récoltées lors de cette expérience nous ont permis, via une analyse basée sur les tests d'hypothèses (Sections 12, 13 et 14) de tirer plusieurs conclusions. L'expérience permet de mettre en évidence l'influence du niveau d'étude sur la façon de réaliser des tâches de maintenance sur des diagrammes de classes UML. Les conclusions qui sont liées à ces résultats sont résumées en Section 15.1.

16.3.2 TAUPE

Le développement de TAUPE (Section 8.2) est une de nos contributions majeures. Ce logiciel permet aux chercheurs en génie logiciel expérimental et en oculométrie de manipuler des données fournies par des oculomètres. Si sa fonction principale est de générer des métriques sur la base de ces données au travers de divers algorithmes, il rend également possible leur visualisation.

TAUPE intervient comme un outil utile au vu du faible nombre de logiciels de ce type. De plus, les autres logiciels existants sont généralement propriétaires et liés à un seul oculomètre. L'architecture hautement évolutive de TAUPE et la licence GPL sous laquelle il est distribué lui permettent d'être facilement maintenable par ses futurs utilisateurs. Cette qualité de maintenabilité correspond à l'ajout et la modification de métriques, d'algorithmes, d'oculomètres ou encore de formats de fichiers de sortie.

16.3.3 Métrique du parcours visuel

L'ensemble des métriques existantes dans le domaine de l'oculométrie subit un manque en terme d'analyse de parcours visuel du point de vue de la "séquence" des fixations. Dans la Section 8.2.3, nous introduisons la notion dynamique de parcours visuel sous une nouvelle métrique nommée "*Distance Relative entre Chemins Visuels*" (cf. Figure 9.1).

À notre connaissance, il n'existe pas de précédents dans l'utilisation d'une distance d'édition pour la quantification de la différence de deux parcours visuels. Cette innovation nous permet de tirer des

conclusions sur les données collectées par Van den Plas [Van09] (voir Chapitre 9), à savoir que les débutants semblent parcourir les diagrammes de façon plus systématique que les experts, ceux-ci utilisant leur expertise afin de déterminer plus rapidement les informations pertinentes d'un diagramme [DLG+11]. Ces conclusions sont complétées par l'application de la métrique aux données récoltées durant notre expérience (Section 13.5). Celles-ci ont en effet montré que les Masters se comportent de façon plus homogène entre eux que les Doctorants lorsqu'il s'agit de parcourir visuellement un diagramme UML. Elles confirment également les observations faites par Yusuf et al. [YKM07].

16.3.4 Travail de synthèse

Pour la rédaction de ce document et afin de réaliser notre expérience, nous avons effectué un travail de synthèse non seulement concernant le patron architectural Modèle-Vue-Contrôleur (et ses variantes, cf. Section 2.4) mais également concernant la maintenabilité des programmes (Section 3) et sciences cognitives en compréhension de programmes (Section 4.1). Ces synthèses regroupent de nombreux travaux dont nous nous sommes inspirés afin de créer notre expérience.

En outre, nous proposons une classification des *offsets* présents sur les résultats enregistrés par les oculomètres et les méthodes permettant de les corriger (lorsque cela est possible) en Section 8.1.3. Cette classification basique est fondée sur les expériences passées effectuées avec les oculomètres du laboratoire *Ptidej* telles que les expériences de Cepeda [CG10], Van den Plas [Van09] et Jeanmart [Jea08].

Glossaire

Anti-patron Un anti-patron (de l'anglais "*anti-pattern*") est une forme littéraire qui décrit une solution fréquente (à un problème) qui génère des conséquences négatives [BMMM98].

Approche GQM Goal-Question-Metric définit une approche de mesure à trois niveaux pour les métriques logicielles. Les trois niveaux sont le niveau conceptuel (le but), le niveau opérationnel (la question) et le niveau quantitatif (la métrique) [BCR94].

Architecture Logicielle L'architecture logicielle d'un système informatique est l'ensemble des structures nécessaires au raisonnement au sujet du programme, ce qui comprend les composants logiciels, les relations entre eux et leurs propriétés [CBB+10].

camelCase Convention de nommage des identificateurs utilisé par certains développeurs. Si plusieurs mots sont liés ensemble pour former un nom, la première lettre des mots imbriqués doit être une majuscule [SB08, p. 6].

Débogage Le débogage est un processus méthodique de recherche et de réduction du nombre de bogues, ou de défauts, dans un programme informatique ou dans du matériel électronique afin de rendre le comportement attendu [Wik11b].

Développeur Une personne qui réalise des activités de développement (incluant l'analyse des exigences, la conception, les tests et l'approbation) pendant le processus de développement logiciel [ISO99].

Diagramme de classes Les diagrammes de classes UML sont la base de la modélisation orientée objet. Les modèles de classes montrent les classes dans un programme, leur inter-relations (en incluant l'héritage, l'aggrégation et l'association) et les opérations et les attributs des classes [Amb04].

Effet coktail party Cet effet décrit la capacité de concentrer son attention auditive sur un seule orateur parmi un ensemble de conversations et de bruits de fond, en ignorant les autres conversations [Wik11a].

Enveloppe convexe L'enveloppe convexe d'un objet ou d'un regroupement d'objets géométriques est l'ensemble convexe le plus petit parmi ceux qui le contiennent.

Fixation Une fixation est une position du regard durant un parcours visuel.

Flux de données Architecture logicielle basée sur l'idée que changer la valeur d'une variable doit automatiquement forcer le recalcul des valeurs des variables qui y sont liées..

Fovéa La fovéa est un zone riche en photo-récepteurs située au centre de la rétine..

Framework logiciel (ou cadriciel) Un *kit* de composants logiciels structurels qui servent à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel [Wik11c].

Homoscédasticité Uniformité de la variance de l'erreur dans un ensemble de valeurs observées.

IDE Un environnement de développement intégré (EDI ou IDE en anglais pour *Integrated Development Environment*) est un programme regroupant un ensemble d'outils pour le développement de logiciels.

Identificateur En sciences informatiques, les identificateurs (IDs) sont des symboles lexicaux qui désignent des entités. Le concept est analogue à celui d'un nom. Les identificateurs sont largement utilisés dans pratiquement tous les systèmes de traitement de l'information. Nommer des entités permet de se référer à eux, ce qui est essentiel pour tout type de traitement symbolique [Wik11d].

Idiome Patron de bas niveau spécifique à un langage de programmation. Il décrit comment implémenter les aspects des composants ou des relations entre eux en utilisant les caractéristiques d'un langage donné [BMR+96] (de l'anglais *idiom*).

Interface Utilisateur Dispositif qui permet à l'usager de manipuler la machine. L'interface utilisateur peut être de plusieurs types : graphique, en ligne de commande ou Web.

Listener En programmation orientée objet, un *listener* est un objet en attente d'un évènement particulier pour lui répondre d'une certaine façon.

Logiciel libre Logiciel respectant les quatre lois suivantes : (0) la liberté d'utiliser le programme comme on le souhaite, pour quelque but que ce soit ; (1) celle d'étudier le code source du programme et de le modifier pour qu'il fasse ce que l'on souhaite ; (2) celle de distribuer des copies du programme afin d'aider son prochain ; (3) celle de distribuer des copies des versions que l'on a modifiées, afin que la communauté entière en bénéficie [Wil02].

Look and Feel En conception logicielle, le *Look and Feel* est un terme utilisé à l'égard des interfaces graphiques utilisateurs et comprend les aspects de sa conception, en incluant les éléments tels que la couleur, les formes, la mise en page et la typographie (le "look"), ainsi que le comportement des éléments dynamiques tels que les boutons [...] et les menus (le "*feel*") [Wik11f].

Maintenabilité La facilité avec laquelle un système logiciel ou un composant peut être modifié pour corriger ses fautes, améliorer sa performance ou d'autres attributs, ou l'adapter à un changement d'environnement.

Méthode (en génie logiciel) Procédure formelle pour la production de résultats [PA09].

Modèle d'application Dans une architecture MVC, le modèle d'application est l'objet qui a connaissance des vues et qui permet aux vues d'obtenir l'information et les notifications..

Modèle du domaine Dans une architecture MVC, le modèle du domaine consiste en objets qui représentent et supportent l'essence du problème [Dea09].

Monitoring (ou surveillance) de l'expérience Dans le cadre de notre expérience, nous appellons "monitoring" l'activité qui consiste notamment à surveiller un sujet, l'accompagner tout au long de l'expérience mais également de vérifier l'état de l'enregistrement de l'oculomètre.

Outil (en génie logiciel) Instrument ou système automatisé pour accomplir quelque chose d'une meilleure façon [PA09].

Paradigme (en génie logiciel) Approche particulière ou philosophie de construction de logiciel [PA09].

Patron de conception En génie logiciel, un patron de conception (*design pattern* en anglais) est une solution générique d'implémentation répondant à un problème spécifique.

Pluggable En informatique, un *plug-in* est un ensemble de composants logiciels qui ajoutent des capacités spécifiques à une application logicielle plus grande [Wik11g]. L'adjectif *pluggable* correspond à la caractéristique d'adaptabilité d'un *plug-in*.

Portage Le portage est le processus d'adaptation logicielle afin qu'un programme exécutable puisse être créé pour un environnement informatique qui est différent de celui pour lequel il est initialement conçu (par exemple, différents CPU, systèmes d'exploitation...). Le terme est également utilisé pour rendre un logiciel utilisable dans différents environnements lorsque le logiciel/matériel est modifié [Wik11h].

Procédure (en génie logiciel) Combinaison d'outils et de techniques qui entrent en concert et produisent un produit particulier [PA09].

Proxy Patron de conception visant à ajouter une indirection à l'utilisation de la classe à laquelle il se substitue.

Résultat significatif En statistiques, un résultat est dit statistiquement significatif lorsqu'il est improbable qu'il puisse être obtenu par un simple hasard.

Scrolling En informatique graphique, en cinéma, télévision et autres affichages dynamiques, le *scrolling* est le défilement de texte, d'images ou de vidéos sur écran [Wik11j].

String Chaîne de caractères (type), en programmation.

Sujet Un sujet est une personne ayant passé notre expérience.

Technique (en génie logiciel) cf. Méthode (en génie logiciel).

Théorie de le Gestalt (en perception visuelle) Les lois de la Gestalt guident les études sur comment nous percevons les composants visuels comme des patrons ou comme un tout. Gestalt est un mot allemand qui signifie "configuration ou patron". Selon cette théorie, il existe six facteurs principaux qui déterminent comment le système visuel groupe les éléments automatiquement en patrons : proximité, similarité, clôture, symétrie, destin commun et continuité [Wik11k].

Zone d'intérêt Une zone d'intérêt est une zone sur une image d'un diagramme. Chaque zone possède un niveau de pertinence. Plus celui-ci est élevé plus nous considérons que la classe dans cette zone d'intérêt est importante pour répondre à la question posée.

Acronymes

CPU Central Processing Unit.
CSV Comma-Separated Values.

DIRO Département d'Informatique et de Recherche Opérationnelle.
DIT Depth Inheritance Tree.
DRCV Distance Relative entre Chemins Visuels.

GIGL Génie Informatique et Génie Logiciel.
GQM Goals - Questions - Metrics.
GUI Graphical User Interface.
GWT Google Web ToolKit.

IEEE Institute of Electrical and Electronics Engineers.
ISO International Organization for Standardization.

JFC Java^{TM} Foundation Classes.

LCS Longest Common Subsequence.
LGPL Lesser General Public Licence.
LOC Lines Of Code.

MW Mental Workload.

NASA National Aeronautics and Space Administration.

PAC Presentation Abstraction Control.
PPSE Partie de Programme Sous Étude.
PReCISE Research Center in Information Systems Engineering.
Ptidej Pattern Trace Identification, Detection, and Enhancement in Java.

RFV Restricted Focus Viewer.

Soccer SOftware Cost-effective Change and Evolution Research.
SSBSE Symposium on Search Based Software Engineering.
SWAT Subjective Workload Asssessment Technique.

TAUPE Thoroughly Analysing the Understanding of Programs through Eyesight.
TLX Task Load Index.

Bibliographie

[AB00] Andy, B. et Blair, M. Twisting the triad : The evolution of the Dolphin Smalltalk MVP application framework. août 2000.

[ACC+07] Aversano, L., Canfora, G., Cerulo, L., Del Grosso, C., et Di Penta, M. An empirical study on the evolution of design patterns. *In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 385–394, New York, NY, USA, 2007. ACM.

[AHU74] Aho, A. V., Hopcroft, J. E., et Ullman, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1^{st} édition, janvier 1974.

[AKGA11] Abbes, M., Khomh, F., Guéhéneuc, Y.-G., et Antoniol, G. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. *In Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering*, CSMR '11, pages 181–190, Washington, DC, USA, 2011. IEEE Computer Society.

[Ale79] Alexander, C. *The Timeless Way of Building*. Oxford University Press, New York, later printing édition, 1979.

[Amb04] Ambler, S. W. *The Object Primer : Agile Model-Driven Development with UML 2.0*. Cambridge University Press, 2004.

[App10] Apple Inc. Cocoa Fundamentals Guide : Cocoa design patterns. `http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html`, décembre 2010. Récupéré le 3 Juillet 2011.

[Bai08] Bailey, R. A. *Design of Comparative Experiments (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge University Press, 2008.

[Bas93] Basili, V. R. The experimental paradigm in software engineering. *In Proceedings of the International Workshop on Experimental Software Engineering Issues : Critical Assessment and Future Directions*, pages 3–12, London, UK, 1993. Springer-Verlag.

[BBK+78] Boehm, B. W., Brown, J., Kaspar, J., Lipow, M., et MacCleod, G. *Characteristics of Software Quality*. North-Holland, Amsterdam, 1978.

[BBvB+01] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., et Thomas, D. Manifesto for agile software development. `http://www.agilemanifesto.org/`, 2001. Récupéré le 1^{er} Août 2011.

[BC87] Beck, K. et Cunningham, W. Using pattern languages for object-oriented programs, septembre 1987. Récupéré le 24 octobre 2010.

[BCM+96] Beck, K., Crocker, R., Meszaros, G., Vlissides, J., Coplien, J. O., Dominick, L., et Paulisch, F. Industrial experience with design patterns. *In Proceedings of the 18th international conference on Software engineering*, ICSE '96, pages 103–114, Washington, DC, USA, 1996. IEEE Computer Society.

[BCR94] Basili, V. R., Caldiera, G., et Rombach, H. D. The goal question metric approach. *In Encyclopedia of Software Engineering*. Wiley, 1994.

[BDLM09] Binkley, D., Davis, M., Lawrie, D., et Morrell, C. To camelCase or Under_score. *In Proceedings of the* 17^{th} *International Conference on Program Comprehension*, pages 158–167, mai 2009.

[BG97] Bellay, B. et Gall, H. A comparison of four reverse engineering tools. *In* Ira BAXTER et Alex QUILICI, éditeurs : *Proceedings of the* 4^{th} *Working Conference on Reverse Engineering*, pages 2–11. IEEE Computer Society Press, octobre 1997.

[BH74] Baddeley, A. D. et Hitch, G. Working memory. *In* Gordon BOWER, éditeur : *The Psychology of Learning and Motivation*, volume 8, pages 47–90. Academic Press, 1974.

[BJ94] Beck, K. et Johnson, R. E. Patterns generate architectures. *In Proceedings of the 8th European Conference on Object-Oriented Programming*, ECOOP '94, pages 139–149, London, UK, 1994. Springer-Verlag.

[BK99] Brilliant, S. S. et Knight, J. C. Empirical research in software engineering : A workshop. *SIGSOFT Softw. Eng. Notes*, 24:44–52, mai 1999.

[BL72] Belady, L. et Lehman, M. An introduction to growth dynamics. *Statistical Computer Performance Evaluation*, 1972.

[BLS06] Briand, L. C., Labiche, Y., et Sauve, A. Guiding the application of design patterns based on UML models. *In Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 234–243, Washington, DC, USA, 2006. IEEE Computer Society.

[BMMM98] Brown, W. J., Malveau, R. C., McCormick, H. W. S., et Mowbray, T. J. *AntiPatterns : Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1998.

[BMR+96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., et Stal, M. *Pattern-Oriented Software Architecture, Volume 1 : A System of Patterns*. Wiley, Chichester, UK, 1996.

[Boe02] Boehm, B. W. *Software Engineering economics*, pages 641–686. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[Boe06] Boehm, B. W. A view of 20th and 21st century software engineering. *In Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 12–29, New York, NY, USA, 2006. ACM.

[Bro78] Brooks, R. *Using a behavioral theory of program comprehension in software engineering*. IEEE Press, Piscataway, NJ, USA, 1978.

[BRZ05] Boehm, B. W., Rombach, H., et Zelkowitz, M. *Foundations of Empirical Software Engineering : The Legacy of Victor R. Basili*. Springer, 2005.

[BT04] Bednarik, R. et Tukiainen, M. *Visual attention tracking during program debugging*, pages 331–334. ACM, New York, NY, USA, 2004.

[BT06] Bednarik, R. et Tukiainen, M. An eye-tracking methodology for characterizing program comprehension processes. *In Proceedings of 5^{th} symposium on Eye Tracking Research & Applications*, pages 125–132. ACM Press, 2006.

[BW08] Boslaugh, S. et Watters, P. *Statistics In A Nutshell*. O'Reilly, 2008.

[Cai07] Cain, B. A review of the Mental Workload literature. *In Defense research and development Toronto Canada*, juillet 2007.

[CBB+10] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., et Stafford, J. *Documenting Software Architectures : Views and Beyond (2nd Edition)*. Addison-Wesley Professional, 2010.

[CCPE09] Cao, A., Chintamani, K. K., Pandya, A. K., et Ellis, R. D. NASA TLX : Software for assessing subjective mental workload. *Behavior Research Methods*, 41(1):113–117, 2009.

[CG10] Cepeda Porras, G. et Guéhéneuc, Y.-G. An empirical study on the efficiency of different design pattern representations in UML class diagrams. *Empirical Software Engineering (EMSE)*, 15(5), janvier 2010.

[CK05] Chabris, C. F. et Kosslyn, S. M. Representational correspondence as a basic principle of diagram design. *In Knowledge and Information Visualization*, pages 36–57. Springer-Verlag, 2005.

[Cli96] Cline, M. P. The pros and cons of adopting and applying design patterns in the real world. *Commun. ACM*, 39:47–49, octobre 1996.

[CS75] Chvatal, V. et Sankoff, D. Longest common subsequences of two random sequences. Rapport technique, Stanford, CA, USA, 1975.

[DCGA08] Di Penta, M., Cerulo, L., Guéhéneuc, Y.-G., et Antoniol, G. An empirical study of the relationships between design pattern roles and class change proneness. *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 217–226, 2008.

[DD09] Deza, M. M. et Deza, E. *Encyclopedia of Distances*. Springer, 2009.

[Dea09] Deacon, J. Model-View-Controller (MVC) Architecture. 2009.

[dFC+11] de Almeida Monte-Mor, J., Ferreira, E. O., Campos, H. F., da Cunha, A. M., et Dias, L. A. V. Applying MDA approach to create graphical user interfaces. *In Information Technology : New Generations (ITNG), 2011 Eighth International Conference on*, pages 766–771, avril 2011.

[DLG+11] De Smet, B., Lempereur, L., Guéhéneuc, Y.-G., Antoniol, G., et Habra, N. Taupe : Visualising and analysing eye-tracking data. *Science of Computer Programming*, Fourth special Issue - Experiment Software and Toolkit, 2011. In Submission.

[DS99] Domsch, M. et Schach, S. A case study in object-oriented maintenance. *In Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*, pages 346–352, 1999.

[Duc07] Duchowski, A. T. Eye tracking methodology. *Theory and Practice*, page 328, 2007.

[Eic03] Eichelberger, H. Nice class diagrams admit good design ? *In* John T. STASKO, éditeur : *Proceedings of the 1^{st} symposium on Software Visualization*, pages 159–168. ACM Press, juin 2003.

[ER03] Endres, A. et Rombach, D. *A Handbook of Software and Systems Engineering*. Addison-Wesley, 1^{st} édition, mars 2003.

[Eye09] Eye Response Technologies Inc. *GazeTracker Reference Manual*, 2009.

[Fac] Seeing Machine's website - FaceLAB. `http://www.seeingmachines.com/product/facelab/`. Récupéré le 23 Decembre 2010.

[FJM50] Fitts, P. M., Jones, R. E., et Milton, J. L. Eye movements of aircraft pilots during instrument-landing approaches. *Aeronautical Engineering Review*, 9(2):24–29, 1950.

[Fow02] Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[Fow04] Fowler, M. Personal Website - Presentation Model. `http://www.martinfowler.com/eaaDev/PresentationModel.html`, juillet 2004. Récupéré le 15 Octobre 2010.

[Fow06] Fowler, M. Personal Website - GUI Architecture. `http://martinfowler.com/eaaDev/uiArchs.html`, octobre 2006. Récupéré le 15 Octobre 2010.

[GB98] Gamma, E. et Beck, K. Test infected : Programmers love writing tests. *Java Report*, 3(7):37–50, juillet 1998.

[GHJV94] Gamma, E., Helm, R., Johnson, R., et Vlissides, J. M. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[GK99] Goldberg, J. H. et Kotval, X. P. Computer interface evaluation using eye movements : Methods and constructs. *International Journal of Industrial Ergonomics*, 24(6):631–645, octobre 1999.

[GOPR01] Genero, M., Olivas, J., Piattini, M., et Romero, F. Using metrics to predict OO information systems maintainability. *In Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, CAiSE '01, pages 388–401, London, UK, UK, 2001. Springer-Verlag.

[Gos05] Gossman, J. Introduction to Model/View/ViewModel pattern for building WPF apps. `http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx`, octobre 2005. Récupéré le 18 Octobre 2010.

[GPC00] Genero, M., Piattini, M., et Calero, C. Early measures for UML class diagrams. 16(4): 489–515, 2000.

[GR83] Goldberg, A. et Robson, D. *Smalltalk-80 : The language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[Gué04] Guéhéneuc, Y.-G. A reverse engineering tool for precise class diagrams. *In* Janice SINGER et Hanan LUTFIYYA, éditeurs : *Proceedings of the* 14^{th} *IBM Centers for Advanced Studies Conference (CASCON)*, pages 28–41. ACM Press, octobre 2004. 14 pages.

[Gué06] Guéhéneuc, Y.-G. Taupe : Towards understanding program comprehension. *In* Hakan ERDOGMUS et Eleni STROULIA, éditeurs : *Proceedings of the* 16^{th} *IBM Centers for Advanced Studies Conference (CASCON)*, pages 1–13. ACM Press, octobre 2006.

[Gué09] Guéhéneuc, Y.-G. A theory of program comprehension — Joining vision science and program comprehension. *International Journal of Software Science and Computational Intelligence (IJSSCI)*, 1(2), Avril–Juin 2009. 47 pages.

[Gus97] Gusfield, D. *Algorithms on strings, trees, and sequences : Computer science and computational biology*. Cambridge University Press, New York, NY, USA, 1997.

[Hab10] Habra, N. INFOM432 – Qualité des produits et des processus – Partie 1, 2010.

[Hal08] Hall, C. PureMVC : Implementation, idioms and best pratices. `http://puremvc.org/component/option,com_wrapper/Itemid,174/`, février 2008. Récupéré le 5 Juillet 2011.

[HCC08] Hsueh, N.-L., Chu, P.-H., et Chu, W. A quantitative approach for evaluating the quality of design patterns. *J. Syst. Softw.*, 81:1430–1439, août 2008.

[HH04] Hadar, I. et Hazzan, O. On the contribution of UML diagrams to software system comprehension. *journal of Object Technology*, 3(1):143–156, Janvier–Février 2004.

[HPS08] Heineman, G. T., Pollice, G., et Selkow, S. *Algorithms in a Nutshell (In a Nutshell (O'Reilly))*. O'Reilly Media, 2008.

[IEE91] IEEE. IEEE standard computer dictionary. A compilation of ieee standard computer glossaries. *IEEE Std 610*, page 1, 1991.

[ISO99] ISO/IEC. ISO/IEC 9126-1 : Software engineering – Product quality – Part 1 : Quality model. Rapport technique, novembre 1999.

[ISO00a] ISO/IEC. ISO/IEC 9000 :2000 Quality management systems – Fundamentals and vocabulary. Rapport technique, International Organization for Standardization, 2000.

[ISO00b] ISO/IEC. ISO/IEC 9126-3 : Software engineering – Product quality – Part 3 : Internal metrics. Rapport technique, décembre 2000.

[Jal08] Jalote, P. *A Concise Introduction to Software Engineering*. Springer Publishing Company, Incorporated, 1 édition, 2008.

[JC76] Just, M. et Carpenter, P. A. Eye fixations and cognitive processes. *Cognitive Psychology*, 8(4):441–480, 1976.

[Jea08] Jeanmart, S. Evaluation de l'impact d'un patron de conception sur la comprehension et la maintenance de programmes – une experimentation par un systeme d'eye-tracking. Mémoire de master, Facultes Universitaires Notre-Dame de la Paix, Namur, Belgium, 2008.

[JFr09] JFreeChart. `http://www.jfree.org/jfreechart/`, 2009. Récupéré le 7 Juillet 2011.

[JK58] Jacob, R. J. K. et Karn, K. S. Commentary on section 4 : Eye tracking in human-computer interaction and usability research : Ready to deliver the promises. *Work*, (1905), 1958.

[JS04] Jørgensen, M. et Sjøberg, D. I. Generalization and theory-building in software engineering research. *In* Stephen LINKMAN, éditeur : *Proceedings of the 8^{th} international conference on Empirical Assessment in Software Engineering*, pages 29–36. IEEE Computer Society Press, mai 2004.

[JW99] Jackson, D. et Waingold, A. Lightweight extraction of object models from bytecode. *In* David GARLAN et Jeff KRAMER, éditeurs : *Proceedings of the 21^{st} International Conference on Software Engineering*, pages 194–202. ACM Press, mai 1999.

[Ker10] Kereki, F. *Essential GWT : Building for the Web with Google Web Toolkit 2 (Developer's Library)*. Addison-Wesley Professional, 2010.

[KP88] Krasner, G. E. et Pope, S. T. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1988.

[Kru96] Kruglinski, D. *Inside Visual C++ (Microsoft Programming Series)*. Microsoft Press, 1996.

[Law08] Law, D. Taligent MVP in interactive statistical graphics. *Comput. Stat.*, 23:487–495, juillet 2008.

[Leh80] Lehman, M. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, septembre 1980.

[LG00] Liskov, B. et Guttag, J. *Program Development in Java : Abstraction, Specification, and Object-Oriented Design*. Addison-Wesley Professional, 2000.

[LG01] Luximon, A. et Goonetilleke, R. S. Simplified subjective workload assessment technique. *Ergonomics*, 44(3):229–243, 2001.

[LG11] LG. LG introduces the world first glasses-free 3D monitor with eye-tracking technology. `http://lg.co.kr/press/lgnews/news/news_view.jsp?press_no=15238`, juillet 2011. Traduction anglaise : `http://www.slashgear.com/lg-cinema-3d-dx2000-display-uses-eye-tracking-for-glasses-free-3d-12164452/` Récupéré le 27 Juillet 2011.

[LP91] Lalonde, W. R. et Pugh, J. R. *Inside Smalltalk (Volume 2)*. Prentice Hall, 1991.

[LS81] Lientz, B. P. et Swanson, E. B. Problems in application software maintenance. *Commun. ACM*, 24:763–769, novembre 1981.

[LVC89] Linton, M. A., Vissides, J. M., et Calder, P. R. Composing user interfaces with InterViews. *Computer*, 22:8–22, février 1989.

[Mar78] Marr, D. Representing visual information. *In* Allen R. HANSON et Edward M. RISEMAN, éditeurs : *Computer Vision Systems*, pages 61–80. Academic Press, 1978.

[Mar82] Marr, D. *Vision : A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt & Company, 1^{st} édition, June 1982.

[MKRv05] Murphy, G. C., Kersten, M., Robillard, M. P., et Čubraniś, D. The emergent structure of development tasks. *In* Andrew P. BLACK, éditeur : *Proceedings of the* 19^{th} *European Conference on Object-Oriented Programming*, pages 33–48. Springer-Verlag, juillet 2005.

[MSD] MSDN Library. Model-View-Presenter pattern. `http://msdn.microsoft.com/en-us/library/ff647543.aspx`. Récupéré le 17 Octobre 2010.

[MW77] McCall, J. et Walters, G. *Factors in Software Quality*, volume 1, 2 & 3. Nat'l Tech.Information Service, 1977.

[NAS03] NASA Human Performance Group. NASA Task Load Index (TLX) : Computerized version. 2003.

[NBD09] Nguyen, V., Boehm, B. W., et Danphitsanuphan, P. Assessing and estimating corrective, enhancive, and reductive maintenance tasks : A controlled experiment. *In Software Engineering Conference, 2009. APSEC '09. Asia-Pacific*, pages 381–388, dec. 2009.

[New73] Newell, A. You can't play 20 questions with nature and win. *In* W.G. CHASE, éditeur : *Visual Information Processing*. Academic Press, 1973.

[NK01] Noda, N. et Kishi, T. Design pattern concerns for software evolution. *In Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, pages 158–161, New York, NY, USA, 2001. ACM.

[NP98] Navarro-Prieto, R. *The Role of Imagery in Program Comprehension : Visual Programming Languages*. Thèse de doctorat, University of Granada, 1998.

[Obj09] Object Management Group. *UML v2.3 Specification*, février 2009.

[OKK07] Ozkaya, I., Kazman, R., et Klein, M. Quality-Attribute based economic valuation of architectural patterns. *In Proceedings of the First International Workshop on The Economics of Software and Computation*, ESC '07, pages 5–, Washington, DC, USA, 2007. IEEE Computer Society.

[OO04] Osborne, J. W. et Overbay, A. The power of outliers (and why researchers should always check for them). *Practical Assessment, Research & Evaluation*, 9(6):1–12, 2004.

[Ora] Oracle. JDK 6 Swing (Java Foundation Classes (JFC))-related APIs and Developer Guides. `http://download.oracle.com/javase/6/docs/technotes/guides/swing/`. Récupéré le 7 Juillet 2011.

[PA09] Pfleeger, S. L. et Atlee, J. M. *Software Engineering : Theory and Practice (4th Edition)*. Prentice Hall, 2009.

[PAC02] Purchase, H. C., Allder, J.-A., et Carrington, D. Graph layout aesthetics in UML diagrams : User preferences. *journal of Graph Algorithms and Applications*, 6(3):255–279, juin 2002.

[Pal99] Palmer, S. E. *Vision Science : Photons to Phenomenology*. The MIT Press, 1^{st} édition, mai 1999.

[PB04] Poole, A. et Ball, L. J. In search of salience : A response time and eye movement analysis of bookmark recognition. *People and Computers XVIII-Design for Life : Proceedings of HCI 2004*, 2004.

[PB06] Poole, A. et Ball, L. J. Eye tracking in Human-Computer interaction and usability research : Current status and future prospects. *Ghaoui, Claude (Ed.). Encyclopedia of Human Computer Interaction*, 2006.

[PBG+08] Pietinen, S., Bednarik, R., Glotova, T., Tenhunen, V., et Tukiainen, M. *A method to study visual attention aspects of collaboration : Eye-tracking pair programmers simultaneously*, pages 39–42. ACM, New York, NY, USA, 2008.

[PJCK97] Pfleeger, S. L., Jeffery, R., Curtis, B., et Kitchenham, B. Status report on software measurement. *IEEE Softw.*, 14:33–43, mars 1997.

[Por08] Porras, G. C. Analyse a l'aide d'oculometres, de techniques de visualisation UML de patrons de conception pour la comprehension de programmes. Mémoire de master, Université de Montreal, Montreal, QC, Canada, 2008.

[Pot96] Potel, M. MVP : Model-View-Presenter - The Taligent Programming Model for C++ and Java. *Taligent, Inc.*, 1996.

[PS00] Privitera, C. M. et Stark, L. W. Algorithms for defining visual regions-of-interest : Comparison with eye fixations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:970–982, 2000.

[PSF95] Ponsoda, V., Scott, D., et Findlay, J. M. A probability vector and transition matrix analysis of eye movements during visual search. *Acta Psychologica*, 88(2):167–185, 1995.

[PULPT02] Prechelt, L., Unger-Lamprecht, B., Philippsen, M., et Tichy, W. F. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *IEEE Trans. Softw. Eng.*, 28:595–606, juin 2002.

[PUT+01] Prechelt, L., Unger, B., Tichy, W., Brossler, P., et Votta, L. A controlled experiment in maintenance : comparing design patterns to simpler solutions. *Software Engineering, IEEE Transactions on*, 27(12):1134–1144, décembre 2001.

[R D10] R Development Core Team. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[Raj02] Rajlich, V. Program comprehension as a learning process. *In* Yingxu WANG, éditeur : *Proceedings of the* 1^{st} *International Conference on Cognitive Informatics*, pages 343–347. IEEE Computer Society Press, août 2002.

[Ree79a] Reenskaug, T. Model-View-Controllers. Rapport technique, Xerox PARC, décembre 1979.

[Ree79b] Reenskaug, T. Thing-Model-View-Editor : An example from a planningsystem. Rapport technique, Xerox PARC, mai 1979.

[Ree10] Reenskaug, T. Reenskaug personal Website. `http://folk.uio.no/trygver/`, juin 2010. Récupéré le 19 Septembre 2010.

[RM03] Rammerstorfer, M. et Mössenböck, H. Data mappings in the Model-View-Controller pattern. 2890:883–901, 2003. 10.1007/978-3-540-39866-0_15.

[Rob79] Robinson, G. Dynamics of the eye and head during movement between displays : A qualitative and quantitative guide for designers. *Human Factors*, 21 (3):343–352, juin 1979.

[RRCL01] Redline, B. C., Redline, C. D., Census, B. O. T., et Lankford, C. P. Eye-movement analysis : A new tool for evaluating the design of visually administered instruments (Paper and Web). *Prevention*, (May), 2001.

[RT06] Richter, A. et Ternaux, M. Les mouvements oculaires. http ://acces.inrp.fr/acces/ressources/neurosciences/vision, 2005-2006. Récupéré le 4 juillet 2011.

[RWL95] Reenskaug, T., Wold, P., et Lehne, O. A. *Working With Objects : The Ooram Software Engineering Method*. Manning Pubns Co, 1995.

[Saw85] Sawyer, K. The mess at the IRS. pages 84–92, octobre 1985.

[SB08] Sierra, K. et Bates, B. *SCJP Sun Certified Programmer for Java 6*. McGraw-Hill Osborne Media, 2008.

[SC88] Siegel, S. et Castellan, N. *Nonparametric statistics for the behavioral sciences*. McGraw–Hill, Inc., 2e édition, 1988.

[Ser88] Serlin, R. C. *Statistical methods for the social and behavioral sciences /*. W.H. Freeman,, New York :, 1988.

[SFM99] Storey, M.-A. D., Fracchia, F. D., et Müller, H. A. Cognitive design elements to support the construction of a mental model during software exploration. *Journal of Systems and Software*, 44(3):171–185, janvier 1999.

[SG00] Salvucci, D. D. et Goldberg, J. H. *Identifying fixations and saccades in eye-tracking protocols*. New York, NY, USA, 2000.

[She09] Shelest, O. Model-View-Controller, Model-View-Presenter, and Model-View-ViewModel design patterns. `http://www.codeproject.com/KB/architecture/MVC_MVP_MVVM_design.aspx`, octobre 2009. Récupéré le 29 Septembre 2010.

[SM10] Sharif, B. et Maletic, J. I. An eye tracking study on camelCase and under_score identifier styles. *International Conference on Program Comprehension*, 0:196–205, Juin–Juillet 2010.

[Smi87] Smith, R. Panel on design methodology. *SIGPLAN Not.*, 23:91–95, janvier 1987.

[Smi09] Smith, J. WPF Apps With the Model-View-ViewModel design pattern. `http://msdn.microsoft.com/en-us/magazine/dd419663.aspx`, février 2009. Récupéré le 18 Octobre 2010.

[Som06] Sommerville, I. *Software Engineering : (Update) (8th Edition)*. Addison Wesley, 2006.

[Spi03] Spinellis, D. *Code Reading : The Open Source Perspective*. Addison Wesley, 1^{st} édition, mai 2003.

[Spr10] SpringSource. Introduction to Spring Web : MVC framework. `http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html`, octobre 2010. Récupéré le 15 Mai 2011.

[SR 06] SR Research Ltd. *EyeLink II User Manual version (07/02/2006)*, Février 2006.

[SSC08] Shic, F., Scassellati, B., et Chawarska, K. The incomplete fixation measure. *In ETRA '08 : Proceedings of the 2008 symposium on Eye tracking research &#38 ; applications*, pages 111–114. ACM, New York, NY, USA, 2008.

[SSL01] Simon, F., Steinbrückner, F., et Lewerentz, C. Metrics based refactoring. *In* Pedro SOUSA et Jürgen EBERT, éditeurs : *Proceedings of the 5^{th} Conference on Software Maintenance and Reengineering*, pages 30–38. IEEE Computer Society Press, mars 2001.

[Sta05] Stanton, N. *Human factors methods : A practical guide for engineering and design*. Ashgate Pub. Co., 2005.

[Sto05] Storey, M.-A. Theories, methods and tools in program comprehension : Past, present and future. May 2005.

[Sun] Sun Microsystems. A Swing architecture overview. `http://java.sun.com/products/jfc/tsc/articles/architecture/`. Récupéré le 19 Septembre 2010.

[Sun02] Sun Microsystems. Sun - Java BluePrints - J2EE patterns - Model-View-Controller. `http://java.sun.com/blueprints/patterns/`, 2002. Récupéré le 15 Octobre 2010.

[SV04] Schlee, M. et Vanderdonckt, J. Generative programming of graphical user interfaces. *In Proceedings of the working conference on Advanced visual interfaces*, AVI '04, pages 403–406, New York, NY, USA, 2004. ACM.

[SW05] Sun, D. et Wong, K. On evaluating the layout of UML class diagrams for program comprehension. *In* James R. CORDY et Harald GALL, éditeurs : *Proceedings of the 13^{th} International Workshop on Program Comprehension*, pages 317–326. IEEE Computer Society Press, mai 2005.

[Swa04] Swartz, F. UI-Model structure. `http://www.leepoint.net/notes-java/GUI/structure/30presentation-model.html`, 2004. Récupéré le 18 Octobre 2010.

[The] The Apache Foundation. Struts2 Website. `http://struts.apache.org/2.x/index.html`. Récupéré le 3 Juillet 2011.

[The95] The Standish Group. Chaos report. *The Standish Group*, 1995.

[UNMiM06] Uwano, H., Nakamura, M., Monden, A., et ichi Matsumoto, K. Analyzing individual performance of source code review using reviewers' eye movement. *In Proceedings of the 2006 symposium on Eye Tracking Research & Applications*, pages 133–140, mars 2006.

[Van09] Van Den Plas, B. La theorie Vision-Comprehension appliquée aux patrons de conception. Mémoire de master, Facultes Universitaires Notre-Dame de la Paix, Namur, Belgium, 2009.

[Viv90] Viviani, P. Eye movements and their role in visual and cognitive processes. *Elsevier Science*, 1990.

[vMV95] von Mayrhauser, A. et Vans, A. M. *Program Comprehension During Software Maintenance and Evolution*, volume 28, pages 44–55. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[WE93] Wierwille, W. W. et Eggemeier, F. T. Recommendations for mental workload measurement in a test and evaluation environment. *Human Factors*, 35(2):263–281, 1993.

[Wen01] Wendorff, P. Assessment of design patterns during software reengineering : Lessons learned from a large commercial project. *In Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, CSMR '01, pages 77–, Washington, DC, USA, 2001. IEEE Computer Society.

[WF74] Wagner, R. A. et Fischer, M. J. The string-to-string correction problem. *J. ACM*, 21:168–173, January 1974.

[Wik07] Wikipedia. Kanizsa triangle — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/File:Kanizsa_triangle.svg`, mars 2007. Récupéré le 10 Juillet 2011.

[Wik08] Wikipedia. Ishihara test — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/File:Ishihara_1.PNG`, mai 2008. Récupéré le 10 Juillet 2011.

[Wik10] Wikipedia. Degré de liberté (statistiques) — Wikipedia, L'encyclopédie libre. `http://fr.wikipedia.org/w/index.php?title=Degr%C3%A9_de_libert%C3%A9_(statistiques)&oldid=59296701`, novembre 2010. Récupéré le 14 Août 2011.

[Wik11a] Wikipedia. Cocktail party effect — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Cocktail_party_effect`, juin 2011. Récupéré le 13 Juillet 2011.

[Wik11b] Wikipedia. Debugging — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Debugging`, juillet 2011. Récupéré le 13 Juillet 2011.

[Wik11c] Wikipedia. Framework — Wikipedia, L'encyclopédie libre. `http://fr.wikipedia.org/wiki/Framework`, juin 2011. Récupéré le 13 Juillet 2011.

[Wik11d] Wikipedia. Identifier — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Identifier`, juin 2011. Récupéré le 13 Juillet 2011.

[Wik11e] Wikipedia. Levenshtein distance — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Levenshtein_distance`, juillet 2011. Récupéré le 14 Juillet 2011.

[Wik11f] Wikipedia. Look and feel — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Look_and_feel`, mars 2011. Récupéré le 13 Juillet 2011.

[Wik11g] Wikipedia. Plug-in (computing) — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Plug-in_(computing)`, juillet 2011. Récupéré le 13 Juillet 2011.

[Wik11h] Wikipedia. Porting — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Porting`, juillet 2011. Récupéré le 13 Juillet 2011.

[Wik11i] Wikipedia. Quartile — Wikipedia, L'encyclopédie libre. `http://fr.wikipedia.org/wiki/Quartile`, mai 2011. Récupéré le 1^{er} Août 2011.

[Wik11j] Wikipedia. Scrolling — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Scrolling`, mai 2011. Récupéré le 13 Juillet 2011.

[Wik11k] Wikipedia. Visual perception — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/Visual_perception`, août 2011. Récupéré le 15 Août 2011.

[Wil02] Williams, S. *Free as in Freedom : Richard Stallman's Crusade for Free Software*. O'Reilly Media, 2002.

[WRH+99] Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C., Regnell, B., et Wesslen, A. *Experimentation in Software Engineering : An Introduction*. Kluwer Academic Publishers, 1st édition, décembre 1999.

[YKM07] Yusuf, S., Kagdi, H., et Maletic, J. I. Assessing the comprehension of UML class diagrams via eye tracking. *In* Eleni STROULIA et Paolo TONELLA, éditeurs : *Proceedings of the 15^{th} International Conference on Program Comprehension*, pages 113–122. IEEE Computer Society Press, juin 2007.

[YWG04] Yi, T., Wu, F., et Gan, C. A comparison of metrics for UML class diagrams. *SIGSOFT Softw. Eng. Notes*, 29:1–6, September 2004.

Index

ANNEXE **A**

Tableaux de données

A.1 Analyse du temps passé

Moyenne	MVC	MVP	MD
Compréhension	113.16193	79.48644	98.23321
Modification	65.28848	65.78126	92.105

TABLE A.1 – Moyennes des temps passés sur les diagrammes en fonction du type de tâche à effectuer et du patron architectural (en secondes)

Moyenne	Swing	JFreeChart
Compréhension	106.63322	77.61513
Modification	100.29756	63.66392

TABLE A.2 – Moyennes des temps passés sur les diagrammes en fonction du type de tâche à effectuer et du framework (en secondes)

A.2 Analyse de la validité des réponses

	MVC	MVP	MD
Compréhension	77,27%	70,83%	65,21%
Modification	63,63%	68,57%	61,29%
Moyenne	69,09%	69,49%	62,96%

TABLE A.3 – Pourcentage de validité des réponses en fonction du patron architectural et du type de tâche effectuée

	Swing	JFreeChart
Compréhension	78,04%	60,07%
Modification	41,86%	82,14%
Moyenne	59,52%	75,00%

TABLE A.4 – Pourcentage de validité des réponses en fonction du *framework* et du type de tâche effectuée

A.3 Analyse des mesures subjectives

A.3.1 Données brutes

Subject	Pattern	Question	Framework	Type	Avgfix	PRF	TNF	DS	DT
Subject01	MVC	Q1	JF	C	224,87	58,50	0,7990	14,9020	0,1772
Subject01	MVC	Q2	JF	C	228,72	2,30	0,0399	9,2157	0,0886
Subject01	MVC	Q3	JF	M	197,78	42,42	0,8614	3,5294	0,0287
Subject01	MVC	Q4	JF	M	179,62	66,67	0,7219	1,5686	0,0132
Subject01	MVP	Q1	S	C	247,07	26,97	0,7847	19,8039	0,1545
Subject01	MVP	Q2	S	C	254,74	45,66	0,6693	26,4706	0,2407
Subject01	MVP	Q5	S	M	234,35	80,20	0,9511	9,0196	0,0982
Subject01	MVP	Q6	S	M	249,01	38,12	0,7877	18,0392	0,1640
Subject02	MD	Q1	S	C	147,21	18,72	0,6848	15,8824	0,1125
Subject02	MD	Q2	S	C	144,24	61,83	0,7803	12,7451	0,0970
Subject02	MVP	Q1	JF	C	120,86	58,30	0,8219	12,7451	0,0994
Subject02	MVP	Q2	JF	C	109,41	79,31	0,9125	4,5098	0,0180
Subject02	MVP	Q3	JF	M	125,25	57,21	0,9011	13,5294	0,1449
Subject02	MVP	Q4	JF	M	141,09	29,63	0,7318	5,6863	0,0718
Subject02	MVP	Q5	JF	M	137,98	18,70	0,2113	11,5686	0,1413
Subject03	MD	Q3	JF	M	183,21	50,00	0,6509	3,9216	0,0216
Subject03	MD	Q4	JF	M	175,73	25,88	0,4963	17,0588	0,1221
Subject03	MD	Q6	JF	M	202,54	69,87	0,9152	25,0980	0,1006
Subject03	MVC	Q1	S	C	173,64	20,92	0,6425	49,2157	0,5544
Subject03	MVC	Q2	S	C	188,87	51,37	0,8141	34,5098	0,3149
Subject03	MVC	Q4	S	C	167,56	65,37	0,8928	39,2157	0,6681
Subject03	MVC	Q5	S	M	194,32	49,82	0,8121	21,7647	0,2897
Subject04	MVC	Q1	JF	C	242,63	34,96	0,5386	34,7059	0,3484
Subject04	MVC	Q2	JF	C	208,85	40,53	0,5380	36,2745	0,3532
Subject04	MVC	Q3	JF	M	248,71	37,36	0,8156	27,6471	0,2586
Subject04	MVC	Q4	JF	M	256,89	28,72	0,5656	12,5490	0,0982
Subject04	MVC	Q5	JF	M	249,42	34,41	0,5743	20,3922	0,1916
Subject04	MVC	Q6	JF	M	221,16	45,70	0,7805	32,7451	0,2191
Subject04	MVP	Q1	S	C	277,67	18,67	0,7209	22,5490	0,1556
Subject04	MVP	Q2	S	C	246,38	62,63	0,8355	37,4510	0,3831
Subject04	MVP	Q4	S	C	220,38	73,33	0,8880	31,9608	0,3484
Subject05	MD	Q1	JF	C	186,36	31,51	0,5849	12,1569	0,0635
Subject05	MD	Q3	JF	M	214,28	54,95	0,7816	12,7451	0,1293
Subject05	MD	Q4	JF	M	175,21	88,10	0,9288	4,3137	0,0359
Subject05	MD	Q5	JF	M	259,12	51,20	0,6359	13,9216	0,1006
Subject05	MVC	Q1	S	C	240,62	65,17	0,9332	9,2157	0,1042
Subject05	MVC	Q2	S	C	276,55	93,33	0,9920	3,3333	0,0335
Subject05	MVC	Q5	S	M	204,05	50,32	0,8198	18,8235	0,0778
Subject06	MD	Q1	S	C	253,39	29,23	0,8225	22,9412	0,2562
Subject06	MD	Q2	S	C	283,96	75,55	0,8739	19,4118	0,2466
Subject06	MD	Q3	S	M	263,33	97,40	0,9686	11,3725	0,0359

Subject06	MD	Q4	S	C	211,22	72,05	0,9059	34,3137	0,4681
Subject06	MD	Q5	S	M	213,23	53,13	0,8275	31,1765	0,1952
Subject06	MVP	Q1	JF	C	193,13	16,59	0,4052	32,1569	0,2634
Subject06	MVP	Q3	JF	M	247,22	60,94	0,8305	12,3529	0,1125
Subject06	MVP	Q4	JF	M	231,84	37,27	0,7967	16,2745	0,1461
Subject06	MVP	Q5	JF	M	191,88	56,63	0,5581	28,8235	0,2802
Subject06	MVP	Q6	JF	M	200,70	35,19	0,7573	23,5294	0,1389
Subject07	MD	Q2	S	C	264,16	48,12	0,5772	15,8824	0,1042
Subject07	MD	Q4	S	C	237,25	78,11	0,8991	22,9412	0,2323
Subject07	MD	Q5	S	M	252,12	71,70	0,8100	27,2549	0,1652
Subject07	MD	Q6	S	M	241,66	90,57	0,9554	18,0392	0,0958
Subject07	MVP	Q3	JF	M	264,68	60,00	0,8987	12,1569	0,0958
Subject07	MVP	Q4	JF	M	241,16	53,18	0,8130	22,5490	0,1640
Subject07	MVP	Q5	JF	M	241,57	70,34	0,7731	23,3333	0,3137
Subject08	MVC	Q1	JF	C	155,86	58,62	0,7605	7,4510	0,0730
Subject08	MVC	Q3	JF	M	133,34	64,29	0,9455	5,2941	0,0551
Subject08	MVC	Q4	JF	M	164,44	54,29	0,7353	11,7647	0,0431
Subject08	MVC	Q5	JF	M	86,79	50,00	0,4884	5,8824	0,0299
Subject08	MVP	Q1	S	C	89,01	37,50	0,9078	8,6275	0,0802
Subject08	MVP	Q2	S	C	180,16	0,00	0,0000	4,9020	0,0144
Subject10	MD	Q1	JF	C	200,09	34,63	0,5714	32,7451	0,2550
Subject10	MD	Q3	JF	M	221,88	61,36	0,9283	13,3333	0,1305
Subject10	MD	Q4	JF	M	257,82	100,00	1,0000	1,7647	0,0168
Subject10	MD	Q5	JF	M	226,09	86,36	0,8772	13,9216	0,3173
Subject10	MD	Q6	JF	M	190,65	65,92	0,9153	25,2941	0,1149
Subject10	MVC	Q1	S	C	271,73	18,22	0,6010	19,0196	0,1616
Subject10	MVC	Q2	S	C	247,08	52,13	0,8717	19,0196	0,2514
Subject10	MVC	Q4	S	C	227,54	93,55	0,9713	26,8627	0,5352
Subject10	MVC	Q5	S	M	241,94	93,10	0,9729	6,2745	0,1078
Subject10	MVC	Q6	S	M	214,05	40,13	0,6689	31,9608	0,3436
Subject11	MVC	Q1	JF	C	197,23	52,53	0,7598	24,5098	0,2155
Subject11	MVC	Q2	JF	C	195,83	27,56	0,4244	16,2745	0,1892
Subject11	MVC	Q3	JF	M	246,90	58,73	0,9202	5,4902	0,0407
Subject11	MVC	Q4	JF	M	221,37	74,67	0,8642	6,4706	0,0850
Subject11	MVC	Q5	JF	M	169,53	55,43	0,6385	20,3922	0,2131
Subject11	MVP	Q1	S	C	195,53	32,11	0,8491	26,4706	0,2502
Subject11	MVP	Q2	S	C	240,39	33,71	0,5469	16,8627	0,1616
Subject11	MVP	Q3	S	M	196,60	55,45	0,7752	33,3333	0,2179
Subject11	MVP	Q4	S	C	261,11	86,87	0,8136	20,9804	0,2227
Subject11	MVP	Q5	S	M	207,80	72,27	0,9440	22,3529	0,2155
Subject11	MVP	Q6	S	M	192,00	66,67	0,9363	7,6471	0,0012
Subject12	MD	Q1	S	C	269,77	29,76	0,8000	27,2549	0,2538
Subject12	MD	Q2	S	C	320,09	54,68	0,7493	27,6471	0,2179
Subject12	MD	Q4	S	C	265,10	65,63	0,8099	36,0784	0,4274
Subject12	MD	Q5	S	M	332,34	29,20	0,7418	14,5098	0,1221

Subject12	MVP	Q1	JF	C	240,68	34,22	0,6871	23,9216	0,1748
Subject12	MVP	Q2	JF	C	203,48	56,42	0,6985	32,5490	0,2071
Subject12	MVP	Q3	JF	M	252,06	47,08	0,7956	25,0980	0,2119
Subject12	MVP	Q4	JF	M	245,62	57,83	0,8446	7,8431	0,0635
Subject12	MVP	Q5	JF	M	204,23	32,92	0,2972	25,2941	0,2778
Subject14	MD	Q1	S	C	250,11	37,27	0,8134	23,7255	0,2419
Subject14	MD	Q2	S	C	234,66	66,67	0,7701	4,3137	0,0419
Subject14	MD	Q5	S	M	257,43	35,90	0,7189	31,9608	0,3293
Subject14	MVP	Q3	JF	M	277,97	49,50	0,7679	10,7843	0,0850
Subject14	MVP	Q4	JF	M	308,33	100,00	1,0000	0,7843	0,0060
Subject14	MVP	Q5	JF	M	359,39	68,18	0,6359	3,5294	0,0419
Subject14	MVP	Q6	JF	M	216,11	28,08	0,5820	14,7059	0,1616
Subject15	MD	Q3	JF	M	185,98	59,46	0,6744	7,0588	0,0335
Subject15	MD	Q6	JF	M	120,03	52,00	0,9153	5,0980	0,0144

TABLE A.6 – Données oculométriques

Subject (Project)	Mental D	Physical D	Temporal D	Effort	Own performance	Frustration	MW	Pattern	Gender	Study
Sujet 1 (Swing)	65%	35%	60%	65%	85%	85%	68,00%	MVP	F	M
Sujet 1 (JFreeChart)	50%	20%	30%	50%	60%	55%	47,67%	MVC	F	M
Sujet 2 (Swing)	80%	15%	65%	70%	75%	60%	72,33%	MD	M	M
Sujet 2 (JFreeChart)	65%	20%	50%	65%	50%	35%	58,00%	MVP	M	M
Sujet 3 (Swing)	70%	40%	70%	60%	55%	35%	63,00%	MVC	M	D
Sujet 3 (JFreeChart)	25%	20%	20%	40%	75%	20%	34,33%	MD	M	D
Sujet 4 (Swing)	100%	0%	0%	100%	50%	0%	66,67%	MVC	F	D
Sujet 5 (JFreeChart)	100%	0%	55%	50%	50%	0%	64,33%	MD	F	D
Sujet 5 (Swing)	75%	0%	55%	85%	0%	95%	65,00%	MVC	F	M
Sujet 5 (JFreeChart)	85%	0%	60%	85%	0%	75%	68,00%	MD	F	M
Sujet 6 (Swing)	20%	0%	20%	10%	55%	15%	21,67%	MD	F	D
Sujet 6 (JFreeChart)	40%	10%	40%	30%	55%	10%	37,33%	MVP	F	D
Sujet 7 (Swing)	30%	10%	10%	20%	60%	0%	25,33%	MD	F	D
Sujet 7 (JFreeChart)	70%	5%	25%	50%	25%	30%	47,00%	MVP	F	D
Sujet 8 (Swing)	65%	0%	55%	50%	25%	70%	54,00%	MVP	F	D
Sujet 8 (JFreeChart)	60%	0%	60%	55%	20%	65%	53,67%	MVC	F	D
Sujet 10 (Swing)	45%	25%	50%	45%	60%	10%	45,67%	MVC	M	M
Sujet 10 (JFreeChart)	65%	35%	60%	65%	55%	15%	59,33%	MD	M	M
Sujet 11 (Swing)	70%	35%	70%	50%	25%	25%	55,67%	MVP	M	M
Sujet 11 (JFreeChart)	50%	60%	40%	50%	45%	65,00%	48,33%	MVC	M	M
Sujet 12 (Swing)	70%	0%	25%	80%	85%	0%	61,00%	MD	M	D
Sujet 12 (JFreeChart)	75%	0%	0%	50%	80%	0%	49,00%	MVP	M	D
Sujet 13 (Swing)	70%	95%	15%	70%	10%	25%	48,00%	MVP	F	D
Sujet 13 (JFreeChart)	90%	70%	45%	65%	10%	25%	59,33%	MVC	F	D
Sujet 14 (Swing)	75%	0%	50%	50%	50%	0%	55,00%	MD	F	D
Sujet 14 (JFreeChart)	60%	0%	35%	55%	45%	0%	47,67%	MVP	F	D
Sujet 15 (Swing)	75%	50%	60%	70%	85%	30%	69,00%	MVC	M	D
Sujet 15 (JFreeChart)	90%	55%	80%	90%	70%	50%	82,67%	MD	M	D
Sujet 16 (Swing)	55%	20%	0%	65%	80%	0%	46,33%	MVC	M	M
Sujet 16 (JFreeChart)	70%	85%	65%	70%	60%	0%	63,00%	MD	M	M

TABLE A.5 – Données NASA-TLX

A.4 Données de comparaison de la similarité visuelle des diagrammes

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
A	0	19	22	0	15	21	88	23	48	35	38	67	135	50	29	39	40	35	30	32	39
B	19	0	29	24	18	14	86	26	59	27	46	93	133	69	10	24	25	22	35	38	38
C	22	29	0	17	24	21	94	6	27	26	50	25	108	46	24	45	46	19	21	49	55
D	0	24	17	0	0	24	96	25	51	32	47	54	133	48	40	32	42	37	32	19	41
E	15	18	24	0	0	17	94	22	65	29	40	82	127	61	15	27	29	9	14	41	45
F	21	14	21	24	17	0	91	24	60	24	40	95	127	72	23	24	34	5	33	29	46
G	88	86	94	96	94	91	0	94	116	85	52	131	99	119	19	40	56	22	35	54	68
H	23	26	6	25	22	24	94	0	72	31	18	103	113	76	25	20	25	32	6	2	45
I	48	59	27	51	65	60	116	72	0	73	54	23	15	45	58	59	64	51	58	68	59
J	35	27	26	32	29	24	85	31	73	0	48	116	103	91	30	17	30	22	29	31	57
K	38	46	50	47	40	40	52	18	54	48	0	90	106	89	18	19	28	24	25	20	65
L	67	93	25	54	82	95	131	103	23	116	90	0	123	4	70	84	82	75	71	77	58
M	135	133	108	133	127	127	99	113	15	103	106	123	0	43	91	84	70	62	64	76	73
N	50	69	46	48	61	72	119	76	45	91	89	4	43	0	72	71	83	70	67	78	27
O	29	10	24	40	15	23	19	25	58	30	18	70	91	72	0	7	12	10	7	17	48
P	39	24	45	32	27	24	40	20	59	17	19	84	84	71	7	0	5	19	13	2	58
Q	40	25	46	42	29	34	56	25	64	30	28	82	70	83	12	5	0	8	7	5	63
R	35	22	19	37	9	5	22	32	51	22	24	75	62	70	10	19	8	0	12	17	52
S	30	35	21	32	14	33	35	6	58	29	25	71	64	67	7	13	7	12	0	0	63
T	32	38	49	19	41	29	54	2	68	31	20	77	76	78	17	2	5	17	0	0	64
U	39	38	55	41	45	46	68	45	59	57	65	58	73	27	48	58	63	52	63	64	0

TABLE A.7 – JFreeChart - Écart-type de la distance entre les classes parmi les différents patrons architecturaux

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	α	β
A	0	0	144	80	3	21	19	29	24	11	19	19	26	4	24	36	27	4	37	15	7	27	26	7	17	28	19	31
B	0	0	167	90	14	10	14	29	16	34	4	19	5	27	21	22	23	25	32	24	33	6	8	27	31	19	21	33
C	144	167	0	5	171	171	169	186	20	66	36	154	166	163	164	162	167	168	155	153	151	145	131	153	49	45	51	43
D	80	90	5	0	96	95	93	83	21	37	18	83	98	96	96	101	97	102	108	100	99	100	78	82	55	50	43	25
E	3	14	171	96	0	15	16	35	28	32	26	8	24	28	5	33	25	5	17	32	33	34	35	39	30	29	18	39
F	21	10	171	95	15	0	16	36	20	26	21	25	5	6	27	21	31	33	24	15	30	23	33	36	38	31	20	41
G	19	14	169	93	16	16	0	28	38	28	9	28	5	14	5	24	19	21	11	18	6	28	24	25	22	31	43	44
H	29	29	186	83	35	36	28	0	54	51	23	47	38	17	25	32	38	36	19	14	11	21	12	22	47	46	45	60
I	24	16	20	21	28	20	38	54	0	18	16	28	37	33	35	45	36	48	53	37	47	47	29	33	28	18	28	32
J	11	34	66	37	32	26	28	51	18	0	12	17	20	19	44	19	47	37	43	28	36	46	39	32	39	38	13	6
K	19	4	36	18	26	21	9	23	16	12	0	32	17	16	16	31	43	44	46	34	21	33	35	29	34	22	36	35
L	19	19	154	83	8	25	28	47	28	17	32	0	37	29	16	33	35	37	29	35	15	25	23	33	22	33	27	6
M	26	5	166	98	24	5	5	38	37	20	17	37	0	15	2	10	9	14	21	24	17	19	22	15	50	43	19	21
N	4	27	163	96	28	6	14	17	33	19	16	29	15	0	0	11	0	16	16	8	7	22	30	19	44	38	34	46
O	24	21	164	96	5	27	5	25	35	44	16	16	2	0	0	8	11	14	1	7	18	5	18	26	30	47	20	20
P	36	22	162	101	33	21	24	32	45	19	31	33	10	11	8	0	0	8	16	21	3	4	19	19	29	40	36	48
Q	27	23	167	97	25	31	19	38	36	47	43	35	9	0	0	11	0	8	19	12	14	16	19	27	29	48	37	49
R	4	25	168	102	5	33	21	36	48	37	44	37	14	16	14	0	8	0	21	21	16	24	29	14	40	50	50	38
S	37	32	155	108	17	24	11	19	53	43	46	29	21	16	1	16	19	21	0	15	12	20	33	5	42	41	38	28
T	15	24	153	100	32	15	18	14	37	28	34	35	24	8	7	21	12	21	15	0	13	20	7	30	49	43	38	50
U	7	33	151	99	33	30	6	11	47	36	21	15	17	7	18	3	14	16	12	13	0	8	27	17	37	38	34	20
V	27	6	145	100	34	23	28	21	47	46	33	25	19	22	5	4	16	24	20	20	8	0	23	13	37	30	44	35
W	26	8	131	78	35	33	24	12	29	39	35	23	22	30	18	19	19	29	33	7	27	23	0	15	33	28	39	46
X	7	27	153	82	39	36	25	22	33	32	29	33	15	19	26	19	27	14	5	30	17	13	15	0	31	41	42	51
Y	17	31	49	55	30	38	22	47	28	39	34	22	50	44	30	29	29	40	42	49	37	37	33	31	0	14	21	20
Z	28	19	45	50	29	31	31	46	18	38	22	33	43	38	47	40	48	50	41	43	38	30	28	41	14	0	27	26
α	19	21	51	43	18	20	43	45	28	13	36	27	19	34	20	36	37	50	38	38	34	44	39	42	21	27	0	2
β	31	33	43	25	39	41	44	60	32	6	35	6	21	46	20	48	49	38	28	50	20	35	46	51	20	26	2	0

TABLE A.8 – Swing/JTable - Écart-type de la distance entre les classes parmi les différents patrons architecturaux

ANNEXE **B**

Publication dans Science Of Computer Programming

À la date du 28 août 2011, la publication suivante a été acceptée pour la quatrième *Special Issue* de *Experimental Software and Toolkits (EST)* du journal *Science of Computer Programming* de *Elsevier*. À cette date, elle est en cours de modification pour révisions **mineures**.

TAUPE: Visualising and Analysing Eye-tracking Data[☆]

Benoît DE SMET[a], Lorent LEMPEREUR[a], Yann-Gaël GUÉHÉNEUC[b], Giuliano ANTONIOL[c], Naji HABRA[a]

[a] *Research Center in Information Systems Engineering, FUNDP, Namur, Belgium*
[b] *Ptidej Team, École Polytechnique de Montréal, Québec, Canada*
[c] *Soccer Lab., École Polytechnique de Montréal, Québec, Canada*

Abstract

Program comprehension is an essential part of any maintenance activity. It allows developers to build mental models of the program before undertaking any change. It has been studied by the research community for many years to devise models and tools to understand and ease this activity. Recently, researchers introduced the use of eye-tracking devices to gather and analyse data about the developers' cognitive processes during program comprehension. However, eye-tracking devices are not completely reliable and, thus, recorded data sometimes must be processed, filtered, or corrected. Moreover, the analysis software tools packaged with eye-tracking devices are not open-source and do not always provide extension points to seamlessly integrate new sophisticated analyses. Consequently, we develop the TAUPE software system to help researchers visualise, analyse and edit the data recorded by eye-tracking devices. The two main objectives of TAUPE are neutrality and extensibility so that researchers can easily (1) apply the system on any eye-tracking data and (2) extend the system with their own analyses. To meet our objectives, we base the development of the TAUPE: (1) on well-known good practices, such as design patterns and a plug-in architecture using reflection, (2) on a thorough documentation, validation and verification process, and (3) on lessons learned from existing analysis software systems. This paper describes the context of development of the TAUPE, the architectural and design choices made during its development, and its documentation, validation and verification process. It also illustrates the application of TAUPE in three experiments on the use of design patterns by developers during program comprehension.

Keywords:
Eye-tracking, Visualisation, Analysis, Neutrality, Extensibility

1. Introduction

Software life-cycle [50] is traditionally divided in several macro phases: from inception to implementation, maintenance, and retirement. The most expensive phase of any software project is maintenance [9]. A software life-cycle often span decades [34] and thus maintenance is rarely performed by the developers that first developed the program. Therefore, maintainers must first understand the program before implementing any change. During program comprehension activity, developers build mental models of the program, which should allow them to perform changes without introducing bugs [49] or degrading the program design [41].

The activity of program comprehension has been studied by many researchers, for example to devise models of program comprehension, *e.g.*, [10, 37, 52], or tools to ease this activity, *e.g.*, [6]. Recently, researchers have introduced the use of eye-tracking devices to further improve our understanding of the developers' cognitive processes taking place during program comprehension [22, 61, 23] by studying their use of visualised data such as source code and identifiers [46], diagrams [53], and so on. In particular, in [23]

[☆] Source code, examples, and documentation are available at http://www.ptidej.net/research/taupe/. This work has been partially funded by the NSERC Research Chair on Software Patterns and Patterns of Software and Guéhéneuc's NSERC Discovery Grant

we proposed a theory to unify previous theories of program comprehension with current theories in vision science [40]. Proposed theory aims to model and explain in details the developers process of acquiring the necessary data using their vision to understand a program.

Eye-tracking devices have been used for nearly 30 years in cognitive science, for the study of human–computer interfaces, for marketing, medical research, and so on. An eye-tracker records the coordinates of a subject's gaze when looking at a computer screen. It provides a new perspective on a subject's comprehension process because it shows the areas attracting the subject's attention as well as the visual path of her gaze on the screen [14]. The subject's attention and visual path together form a window on her cognitive processes [45]. Thus, analysing the data recorded using an eye-tracking device allows understanding in details a subject's process of acquiring data, for example during program comprehension.

However, there are still some major obstacles to the widespread adoption of eye-tracking devices. Besides the costs of such devices, the accompanying analysis software tools are not open-source and often not extensible, preventing the development and seamless integration of new sophisticated analyses [29]. Consequently, we undertook the development of the Taupe system[1] (Thoroughly Analysing the Understanding of Programs through Eyesight) visualise, analyse and edit eye-tracking data. The two main objectives of Taupe are neutrality and extensibility so that researchers can easily (1) apply the system on any eye-tracking data and (2) extend the system with their own analyses. To meet our objectives, we base the development of Taupe: (1) on well-known good practices, such as design patterns and a plug-in architecture using reflection, (2) on a thorough documentation, validation and verification process, and (3) on lessons learned from existing analysis software systems.

In Section 2, this paper describes the context of development of Taupe. In Section 3, it details the architectural and design choices made during its development, and its documentation, validation and verification process. In Section 4, it illustrates the application of Taupe in three experiments on the use of design patterns by developers during program comprehension. Finally, in Section 5, it summarises the contributions of Taupe and concludes with future work.

2. Context

In this section, we first introduce definitions needed to understand the rest of the paper. We then discuss previous work on program comprehension and on using eye-tracking data. Finally, we describe two eye-trackers to show their usage and describe the recorded data that they provide.

2.1. Definitions

In vision science, a *fixation* is the position of the eye during a gaze and a *saccade* is a movement of the eye between two fixations [40]. In the field of empirical software engineering [26] and in particular in this paper, an *experiment* is a set of subjects who answer a set of questions (*e.g.*, comprehension questions) using some visual data (*e.g.*, a UML class diagram). An *area of interest* is an area of some visual data with a specified relevance. An area can thus be *relevant* to the subject to answer a question, *irrelevant*, or *ignorable*. The fixations in an ignorable area of interest should not be taken into account in any subsequent analyses, for example because they pertain to the area of the visual data in which the question to answer was written and, thus, do not provide any interesting data about the subject's cognitive process.

2.2. State of the Art

The Taupe system is used to edit, visualise, and analyse data gather using eye-tracking system during empirical studies of program comprehension activities. We summarise the theories in vision science, report some works on empirical studies in software engineering, and detail studies in program comprehension, emphasising studies on the developers' use of UML class diagrams, which form the main bulk of the eye-tracking studies in software engineering.

[1] "Taupe" means "mole" in French and is pronounced 'tOp.

2.2.1. Theories in Vision Science

Vision science is the domain of computing science interested in the understanding of people's vision system. Vision science collects facts on vision, formulate laws from these facts, and devise theories explaining these laws and facts. With these theories, vision scientists have been able to predict new facts successfully and to refine their theories.

Vision science possesses many theories to explain colour vision, spatial vision, perception of motion and events, as well as eye movements, visual memory, and visual awareness. To the best of our knowledge Palmer's book presents the most complete and in-depth coverage of vision theories, cast in the information processing paradigm [40].

2.2.2. Theories in Software Engineering

The domain of software engineering possesses theories, which are an invaluable help in setting up experiments to observe facts (dis)proving the laws and theories. However, existing theories [16] do not explain well known facts, due to the relatively recent existence of the domain, the complexity of the phenomena, and the lack of agreed upon formalisms and notations. A theory helps in understanding a domain by explaining a list of "Why is it so?" questions [16, p. 7] [39].

Few theories of program comprehension have been proposed in the literature. One of the first theory of program comprehension, proposed by Brooks [11], describes program comprehension as a process of building a sequence of knowledge domains, bridging the gap between problem domain and program execution. A succession of knowledge domains describes a software engineer's comprehension of a program.

Another theory, developed by von Mayrhauser [36], is an integrated theory describing the processes taking place in the software engineers' minds during program comprehension, as a combination of top-down and bottom-up comprehension processes, working with a common knowledge base. This integrated theory accounts for the dynamics of forming and of abstracting a mental representation of a program.

Empirical studies are essential to understand phenomena with which software engineering deals, thanks to the work of precursors, such as Vic Basili [8]. Empirical studies are based on the classical cycle: observations (facts), laws, theories. Many facts have been recorded through empirical studies and some laws have been proposed [34]. Yet, few theories have been derived to explain these facts and laws and to predict news facts. Some researchers argue that current empirical studies focus too much on generalisation and not enough on theory building [32].

2.2.3. Studies of Program Comprehension

The rich literature on program comprehension focused early on the problems of obtaining data from software artifacts (static and dynamic data, features, documentation and other repositories), see for example [4, 51]. It also tackles the means to represent and to communicate this data, using various techniques from text-based editors to 3D interactive dynamic environments, such as [48]. This literature is essential to understand what kind of data software engineers have at their disposal to comprehend programs.

Some works also study the contexts in which the program comprehension activity takes place. Murphy *et al.* [38] distinguish, describe, and identify recurring patterns in the software engineers daily activities. Although not related to program comprehension explicitly, their work brings insight in the program comprehension activity, because program comprehension is part of all but the most basic software engineering activities. Thus, this line of research is important to generalise claims in program comprehension.

All developers use diagrams as a means to convey information to other developers or to better understand programs. Diagrams reduce comprehension and learning effort by omitting irrelevant details and highlighting pertinent information. The closer the information presented on diagrams is to the developer's mental representation, the easier it is to understand [12].

Class diagrams have been extensively studied in the literature on program comprehension. They represent the structure and global behaviour [28] of programs, showing classes, interfaces, and their relationships. They are often used by software engineers during development and maintenance to abstract implementation details and to present an easier-to-grasp clustered view of the program source code [21, 44, 55]. We only recall here some of the main lines of research on program comprehension using class diagrams.

Purchase *et al.* [13] reported the results of experiments on the effect of aesthetics criteria on the preferences of users for UML class diagrams. They performed several experiments with subjects to assess the subjects' preferences over several pairs of class diagrams, each diagram in a pair conforming to different (but related) aesthetics criteria. They collected quantitative data in the form of percentages of subjects preferring one diagram over another in each pair and qualitative assessments of each diagram. They reported the most important aesthetic criteria for UML class diagrams, *e.g.*, joined inheritance arcs or directional indicators.

Eichelberger [15] studied the relation between the UML notation for class diagrams, principles of human–computer interactions, and principles of object-oriented design and programming. The author then suggested changes to the UML notation and aesthetics criteria to lay out class diagrams. These changes and aesthetic criteria are implemented in a tool, SUGIBIB, to lay out UML-like class diagrams. The author claimed that laying out class diagrams conforming to aesthetics criteria improve the readability of the diagrams but only qualitative arguments were provided.

Hadar and Hazzan [25] presented results from a study on the startegies applied by software engineers in the process of comprehending visual models of programs. They use visual models that were described using the UML notation and included use case, activity, class, sequence, collaboration, state chart, object, package, and deployment diagrams. The subjects were senior students majoring in computer science from several universities. The subjects were divided in two groups to collect both qualitative and quantitative data. The authors concluded on the usefulness of multifaceted descriptions of programs provided by the UML and that no one type of diagram was more important than the other. However, further studies should be performed to confirm these findings.

Sun and Wong [55] evaluated the layout algorithms for class diagrams of two industrial tools, Rational Rose and Borland Together, according to criteria from previous work, including the cited works by Purchase [13] and Eichelberger [15]. Using laws from the Gestalt theory of visual perception [40, p. 50–53], they retained and justified 14 criteria to assess the visual quality of the layouts of class diagrams. They applied these criteria on a Thermometer program and on JUNIT [19]. They concluded on the good quality of both industrial tools, on the relevance of their criteria, and on the difficulty of satisfying all criteria.

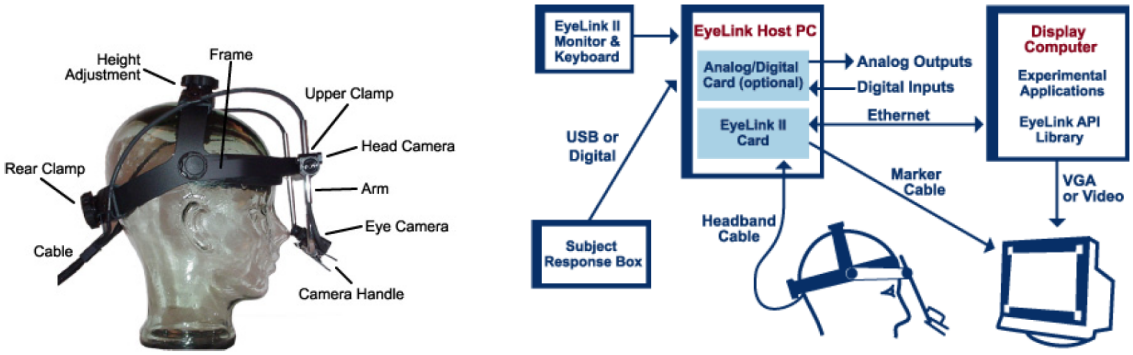
Guéhéneuc also conducted an experiment with eye-trackers to study how software engineers acquire and use information from UML class diagrams [22]. He concluded on the importance of classes and interfaces and reported that developers seem to barely use binary class relationships, such as heritage or composition. Yusuf *et al.* [62] conducted a similar study to analyse the utilization of specific characteristics of UML class diagrams (*e.g.*, layout, color, and stereotypes) during program comprehension. They concluded on the efficiency of layouts with additional information as colors or stereotypes to improve program comprehension.

Bednarik and Tukiainen [5] proposed an approach to study trends on repeated measures of sparse data over a small data set of program comprehension activities captured with eye-trackers. Using this approach, they characterized program comprehension strategies using different program representations (code lecture and program execution). Sharif *et al.* [47] similarly studied whether the Camel Case convention of creating identifiers was more efficient than using underscores. They replicated a previous study by Binkley *et al.* [7] that showed that the Camel Case convention leads to a higher accuracy in reading. Interestingly, they found opposite results: although the data indicated no difference in accuracy between the Camel Case and underscore conventions, subjects recognised identifiers with underscores style more efficiently.

Uwano *et al.* [58] studied the code-reading habits of developers and identified typical "patterns" that distinguishes "efficient" readers from others. They first implemented an integrated environment to measure and record code reviewers' eye movements and, based on their fixations, identify the line of the source code that the reviewers are reading. They then conducted an empirical study of 30 review processes of six programs by five reviewers. They reported that all reviewer "scanned" the source code following a similar pattern and that reviewers who did not spend enough time during the "scan" tended to take more time for finding defects in the code.

2.3. Devices

We now present two eye-tracking devices that are instances of the two main classes of such devices: intrusive (require subjects to wear some gears) and non-intrusive.



(a) Headband (b) Setting

Figure 1: Eye-link II [2]

Figure 2: The FaceLAB eye-tracker [1]

2.3.1. Eye-link II

The Eye-link II system consists of three miniature cameras mounted on a padded headband (see FIGURE 1a). Two eye cameras are used to track the eyes' movements. An optical head-tracking camera integrated into the headband allows an accurate tracking of the subject's point of gaze.

FIGURE 1b from *SR-Research*[2] [2] describes the configuration of this eye-tracking system, which uses two separate computers: the *Host PC* and the *Display PC*. The *Host PC* performs real-time eye-tracking at 250 or 500 samples per second and also computes the subject's true gaze position on the display. This data, stored in a data file on the *Host PC*, is the data used by the TAUPE parser.

The *Desktop PC* provides displays for experiment, and calibrating targets during eye-tracker calibration. Sample display programs, C source code, and instructions for creating experimental programs are provided in the *Eye-link II Windows developer kit* but the `Data Viewer`[3] is neither *open-source* nor *free* and cannot be easily extended with new analyses.

The major problem of this system was its instability. If the subject moved too far away from the calibration point, the calibration step has to be done again. An offset (a difference between the real place of the fixation and the fixation recorded by the eye-tracker device) was present for every subject who took part in an experiment.

2.3.2. FaceLAB and GazeTracker

FaceLAB (from *Seeing Machine*[4]) is a more recent eye-tracking device than the Eye-link II system. It consists of one computer (either a desktop PC or a laptop) and two miniature cameras, as shown in FIGURE 2. It tracks first the position of the head using the eye-brows, the nose, and the lips and, then, uses this

[2] http://www.sr-research.com/accessories_ELII_dv.html
[3] http://www.sr-research.com/accessories_EL1000_dv.html
[4] http://www.seeingmachines.com/

data to track the subject's gaze. It works in pair with *GazeTracker* (from *EyeResponse*[5]), which is a more recent [3] data visualisation software than the `Eye-link II Data Viewer` but which is also neither *open source* nor *free*. Multiple eye-trackers can interact with one another: such a configuration is used in cockpits or in cars to track the eyes of the subject even if she turns her head. It is also possible to run *FaceLAB* and GazeTracker on a single computer with a *scene camera* that records a video of the whole scene.

FaceLAB interacts with the eye-tracking device and transmits the data to *GazeTracker* via the network. *GazeTracker* simply saves the data associated with the diagram's image. *GazeTracker* provides a more advanced interface and more visualisation tools than `Eye-link II Data Viewer` but it cannot be readily extended with new analyses. A screencast of the use of *FaceLAB* and *GazeTracker* is available on-line[6].

3. Taupe

The Taupe system is a software program designed by the *Ptidej Team*[7] to import data from eye-trackers and to enable the execution of various algorithms on the imported data. Its first version has been developed since 2005 with the contribution of many students. Its current version, v2.0, was developed by Benoit De Smet and Lorent Lempereur reusing some code from the first version but revising entirely the program architecture and design. The user and developers' guides[8] provide more details about the versions of Taupe.

3.1. Motivation

Taupe was originally designed to compare the ways that subjects' read and understand UML class diagrams. A subject was asked different questions about her understanding of some diagrams when performing some maintenance tasks. The subjects' eye movements were recorded using an eye-tracking device.

The only software tool able of processing and analysing the eye-tracking recorded data were the proprietary systems provided by the manufacturers of the devices and packaged with their eye-trackers. These systems are neither open-source nor free.

The first version of Taupe had for objectives to ease the analysis of eye-tracking data when subjects must understand UML class diagrams or any other kind of diagrams, to be open-source and free, and to be neutral and extensible.

3.2. Architecture

figure 3 shows the architecture of Taupe v2.0. The core of Taupe consists of the data collected about an experiment: fixations, saccades, questions and their answers. Some of this data is provided by an eye-tracking device. Other data is collected but the experimenters through questionnaires or other means.

The data is organised depending on the answers to the questions. The questions are represented by a set of images and their corresponding area of interest. Each file that contains the areas of interest related to a question can be manually written and Taupe also allows the user to create this kind of files using a graphical user interface. The different types of parsers are used to extract the information from the files provided by the eye-tracking devices.

The possibility to link a subject to other subjects is also available in Taupe, through the concept of *groups*. A group is a set of subjects who have an attribute in common. For example, such an attribute could be the level of study (Bacc., M.Sc., Ph.D., and so on), their gender (male or female), their UML knowledge (low, average, high, and so on). All these variables can be measured by some external means (questionnaires, interviews, and so on).

Taupe handles one experiment at a time. The system's main features are *commands*. For example, the *AOI Maker*, command helps to graphically designate and create areas of interest for a given image. Other commands provides the basic role feature of the system, for example the set of algorithms that produces

[5] http://www.eyeresponse.com/
[6] http://www.ptidej.net/research/taupe/videos/
[7] http://www.ptidej.net/
[8] http://www.ptidej.net/research/taupe/downloads/

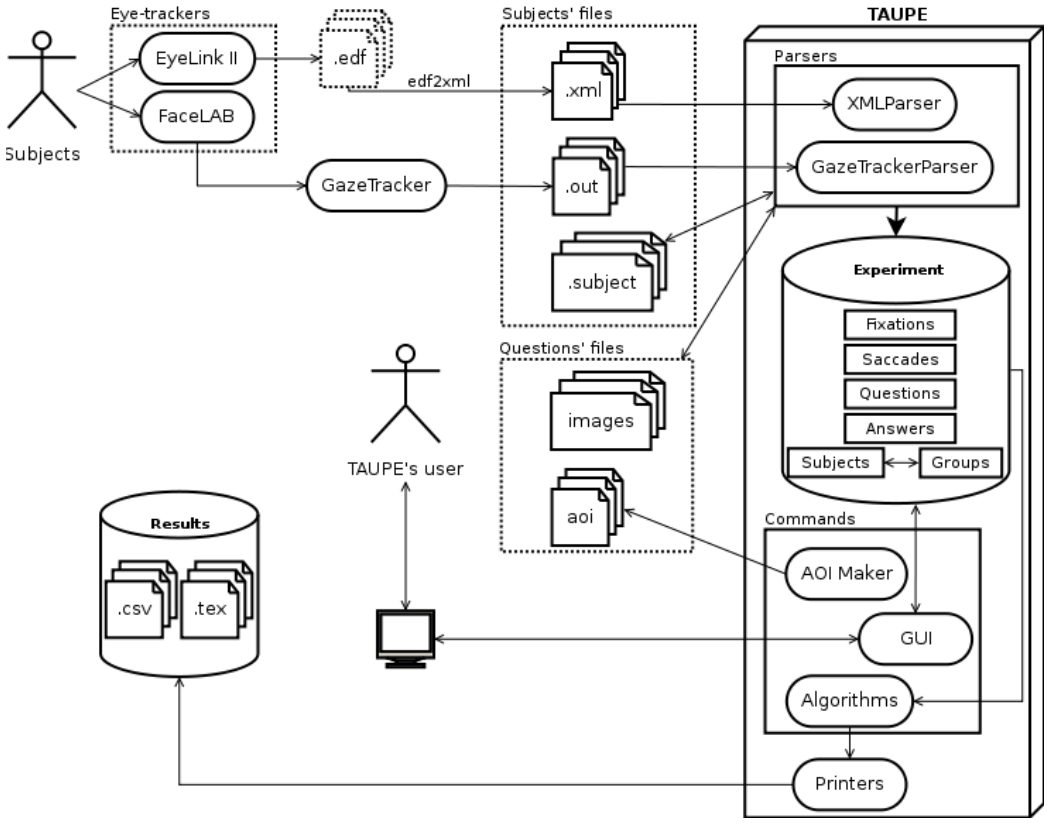


Figure 3: Architecture

analysis results. These results may be written in a set of files using some *printers* selected by the user and they can be subsequently studied or analysed using any data mining software. TAUPE also implements a graphical user interface to visualise data and their characteristics.

3.3. Inspirations

The development of TAUPE has been inspired by many sources.

3.3.1. From the Eye-Trackers

The three main features result from our past uses of eye-tracker devices and their analysis software systems. First, TAUPE must implement some parsers to use the data from multiple eye-tracking devices. Second, TAUPE must be able to graphically display the fixations, saccades, and visual path over a diagram. Third, TAUPE must allow its users to correct the data recorded by the devices using some offsets, as shown in FIGURE 4, which is an example of a diagram with a clearly visible static offset. Every cloud of points must be moved to the same direction and the same distance to match with the diagram.

Three types of offset were encountered during our previous experiments: *static* offsets, *non-static* offsets, and *chaotic* offsets. *Static* offset can be easily seen and a constant translation can be applied to all the fixations to fix the offset. *Non-static* offsets are less easily seen and the translations are different for each group of fixations according to their position on the screen. *Chaotic* offsets are random offsets due to vagaries from the eye-tracking devices and–or the subject and must be corrected manually for each fixations.

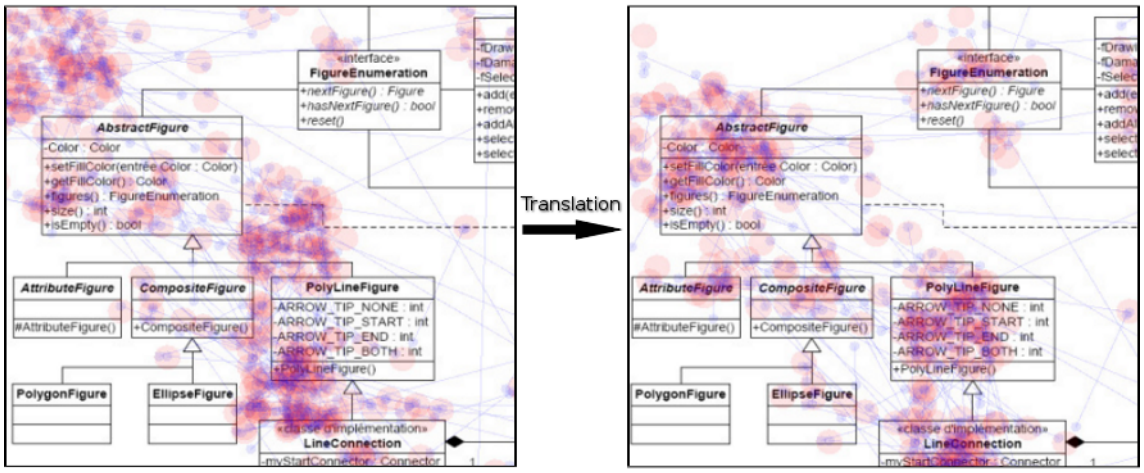


Figure 4: An example of a static offset

3.3.2. From the Fields of Study

The field of study using eye-tracking devices led to several useful metrics that allow to assess a subject's browsing effort, for example. Metrics based on fixations are the most common; for example, the total number of fixations is pointed out by Goldberg *et al.* [20]: "The number of fixations overall is thought to be negatively correlated with search efficiency". The number of fixations per areas of interest indicate that certain areas are more noticeable or more important than other areas [42]. According to Just *et al.* [33], the duration of the fixations on a specific area can have two different meanings:

1. The subject has a hard time to extract the information [18].
2. The subject is "more engag[ed] in some way" by the object [43].

The *spatial density* of fixations is a widely used metric: the "coverage of an interface due to search and processing may be captured by the spatial distribution of gazepoint samples. [...] The interface can be divided into grid areas either representing specific objects or physical screen area. [...] The *spatial density* index was equal to the number of cells containing at least one sample, divided by the total number of grid cells. [...] A smaller spatial density indicated more directed search, regardless of the temporal gazepoint sampling order" [20]. The cell's size used in Taupe is 64×64 pixels.

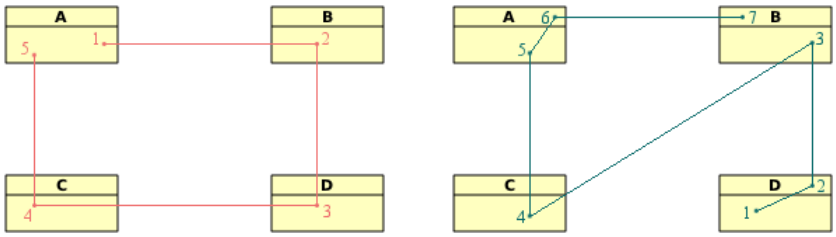
Some metrics use saccades. However, some eye-tracking devices do not provide the required raw data. For example, *FaceLAB* does not provide the *amplitude* of the subject's saccades. Yet, other saccade-based metrics are implemented in Taupe, for example using a *transitional matrix* and its *density*: "also known as link analysis, frequent transitions from one region of a display to another indicates inefficient scanning with extensive search. [...] The transition matrix is a tabular representation of the number of transitions to and from each defined area" [20]. A cell is filled if a saccade starts or ends in its area. The density of the *transitional matrix* can be computed as:

$$TRANSITION\_DENSITY = \frac{\sum_{x \in C} isFilled(x)}{\#C}$$

where C is the set of cells in the *transitional matrix* and $isFilled : C \to \{0, 1\}$ returns 1 if the specified cell is filled and 0 otherwise. Two visual paths can have the same *convex hull* and the same *spatial density* but different *transitional densities*. A better visual path would have a less dense *transitional matrix*.

3.3.3. From Previous Experiments

Taupe is able to compute several statistics about the time spent on each question for each group of subjects. These metrics do not come from the field of eye-tracking but are useful to compare the subjects'



(a) ABDC (b) DBCAB

Figure 5: Two distinct visual paths

performance. Thus, it provides a command that provide statistics about the fixations' duration, the subjects' time spent in a specified area of interest, the transitional matrix. The whole set of metrics and algorithms implemented and available currently in TAUPE are explained in the user's guide[8].

A *visual path* is a series of visited areas of interest, sorted by chronological order. An area is visited if there is a fixation in it. A same area of interest that is visited twice consecutively is considered only once. For example, FIGURE 5 shows two distinct visual paths on a same diagram, with four areas of interest: `A`, `B`, `C`, and `D`. One subjects' visual path, on the sub-figure of the left, is `ABDC` while the other, on the sub-figure of the right, is `DBCAB`. These two visual paths show that the two subjects understood the diagram differently.

The difference between two *visual paths* is computed in TAUPE using an *edit distance* algorithm, the *Levenshtein algorithm* [24], which compares two strings. Using TAUPE, the user can also choose a specified percentage of keeped fixations to generate these *visual paths*. A *merged fixation* is actually all the consecutive fixations that are in an area of interest without leaving this area (the duration of this resulting fixation is the sum of all fixations' duration). The kept percentage of fixations is related to the longer *merged fixations*.

Jeanmart [30] suggested the metric *Normalized Fixations per Area of Interest* that is the ratio between the normalised number of fixations in an area of relevant interest and the normalised number of fixations in an area of irrelevant interest. The $NORM\_RATE_i$ metric can be used to assess a subject's effort:

$$NORM\_RATE_i = \frac{\frac{\#FAORI_i}{\#AORI_j}}{\frac{\#FAOII_i}{\#AOII_j}}$$

where A_j is the set of answers related to the question j, $FAORI_i$ is the set of fixations contained in an area of relevant interest for the subject's answer i, $AORI_j$ is the set of areas of relevant interest in the question j, $FAOII_i$ is the set of fixations contained in an area of irrelevant interest for the subject's answer i, $AOII_j$ is the set of areas of irrelevant interest in the question j, and the # returns the cardinality of a set.

3.4. TAUPE *Use*

FIGURE 6 shows the main user interface of TAUPE.

3.4.1. Areas of Interest

As mentioned in Section 3.2, each question is related to a file that contains its set of areas of interest. Although a user can create such a file using TAUPE's *AOIMaker* command, shown in FIGURE 7; they can also write such a file manually. They must respect the EBNF grammar [27] below:

```
[<id> <type> <name> <coordonate> <coordonate> <coordonate>+ <EOL>]*
<id> ::= integer
<type> ::= NULL | AOI | AORI
<coordonate> ::= "("integer "," integer")"
<EOL> ::= EndOfLine
```

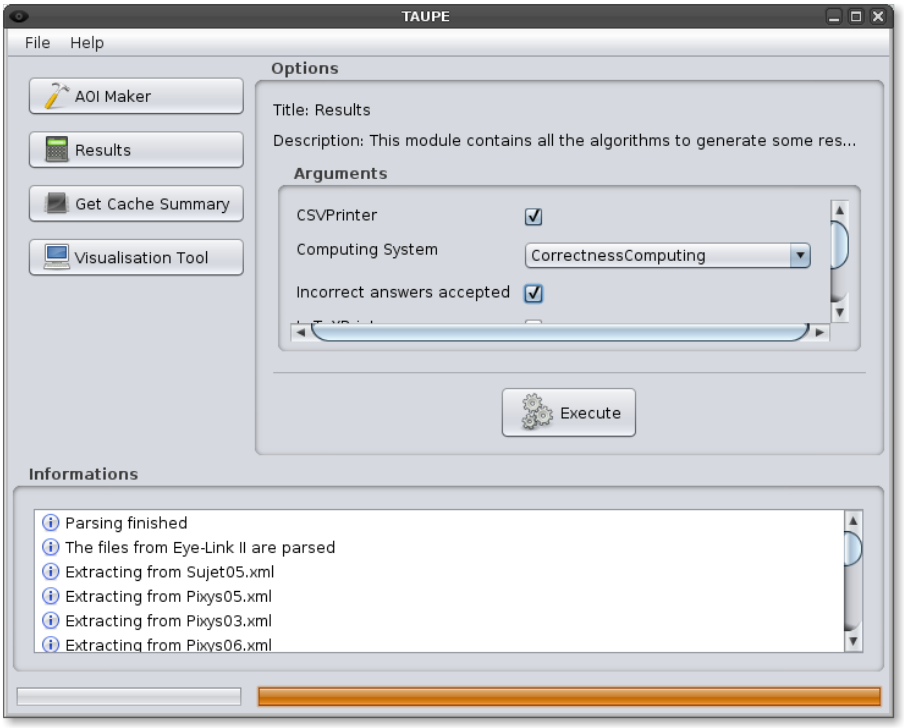


Figure 6: The TAUPE main interface

3.4.2. Visualisation

Although the other analysis software systems in the field of eye-tracking allow user to visualise set of data recorded by these devices, TAUPE brings an innovation in the visualisation of the data and of the analyses performed on the data, for example, the visualisation interface can display, in addition to fixations and saccades, the convex hull of a chosen percentage of fixations, as shown in FIGURE 8.

3.4.3. Input Files

An input file to TAUPE is simply a set of fixations and saccades listed in a text file. Different eye-tracking devices provide different input formats but with essentially the same content: a time (sometimes a *timestamp* in milliseconds), a type (fixation or saccade), a et of coordinates (x and y), and a duration. Other pieces of data, such as the subjects' pupil sizes or the saccades' peak velocity, can also appear in an input file.

Eye-link II : The *SR-Research*'s software system generates `.edf` files that TAUPE cannot read directly. A tool named `edf2xml` (provided by the same company) can be used to generate `.xml` files from the `.edf` files. Then, these `.xml` files can be loaded in TAUPE, which include the saccades' *peak velocity* and *amplitude*.

GazeTracker : The *GazeTracker* software system can generate `.out` files that can be loaded in TAUPE using the appropriate parser. A screencast on how to export data from *GazeTracker* is available on-line[6].

3.5. TAUPE Development

The main objectives of the TAUPE software system are neutrality and extensibility. During the development of its version 2.0, decisions were taken to satisfy our two objectives.

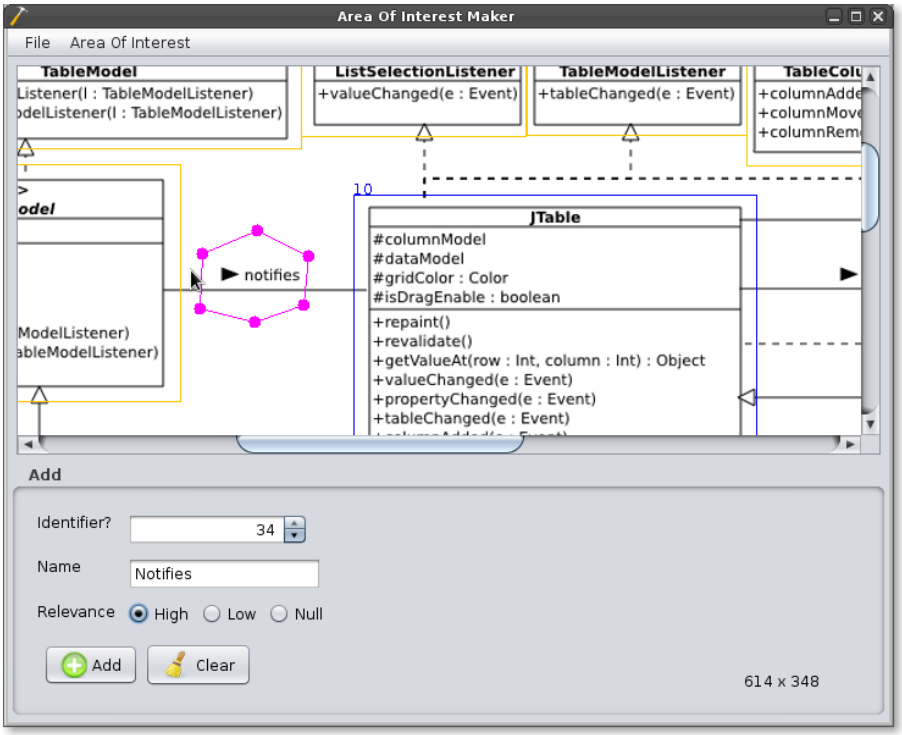


Figure 7: Areas of Interest Maker

3.5.1. Neutrality

The neutrality of the TAUPE system with respect to the eye-tracking devices is achieved using an open architecture in which each core components of the system, including parsers, experiments, commands, and printers, as shown in FIGURE 3, can be specialised or new such component can be added to the system. We achieve such an open architecture using several design patterns and Java reflection mechanism.

First, TAUPE implements the architectural pattern *Model View Controller* (MVC) in its implementation for its user interface, as described by Gamma *et al.* [17]. The MVC uses the *Observer* design pattern to facilitate the communication between the *model* and its *views*.

The *Composite* and *Visitor* design patterns are implemented to represent and visit the data in TAUPE: fixations, saccades, questions, and so on. Results from commands also implement the *Composite* and *Visitor* design patterns to allow hierarchical output and the combination of the outputs of several commands. The printers, which generate output files that contain the results, are a set of results visitors, which define how to print each result (using some `visit` methods) while the results describe how to navigate their hierarchy (using some `accept` methods).

The TAUPE system handles one experiment at a time and, therefore, the `Experiment` class implements the *Singleton* design pattern. It also only returns *Iterator*s on lists and sets as a defensive measure against user code modifying unwillingly or maliciously the content of the experiment or of the results.

3.5.2. Extensibility

Extensibility is one of the two main objectives of the TAUPE system. It pertains to maintainability [57]: "The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment".

We achieve extensibility in TAUPE by following two complementary directions: one related to its implementation and the use of the Java reflection mechanism, another related to its documentation.

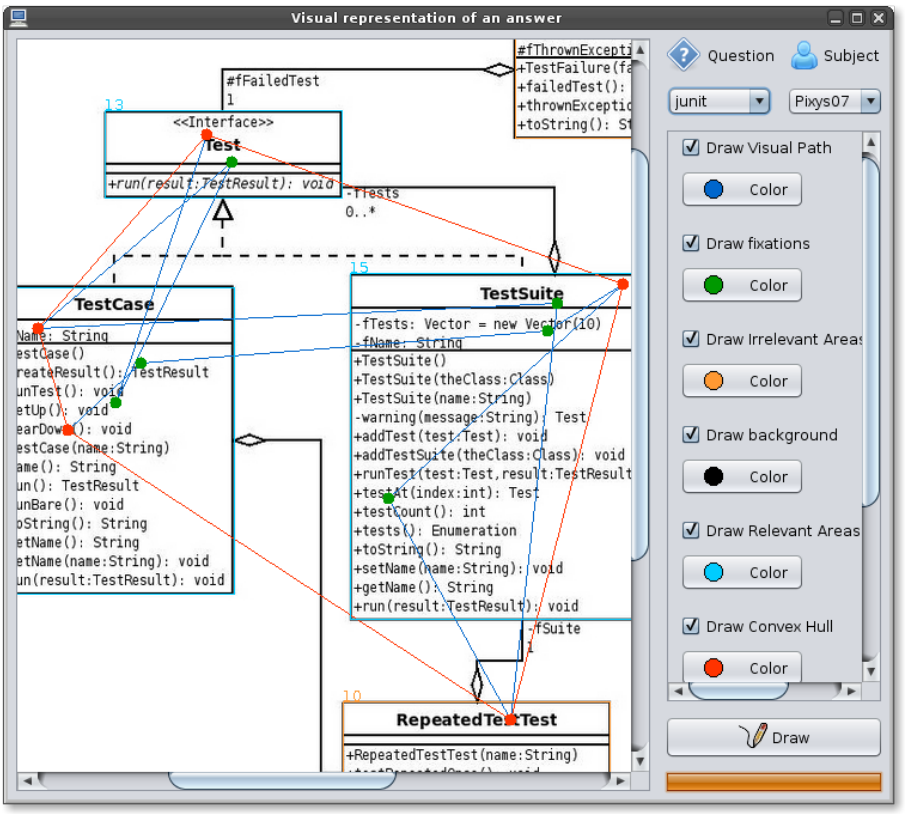


Figure 8: The visualisation tool

First, Taupe makes an extensive use of the Java reflection mechanism to increase its extensibility as described in the following Section 3.5.2 by allowing other developers to easily add their own parsers, commands, and printers through the implementation of the appropriate interfaces and the integration of their implementation is specific folders. New commands (such as new algorithms, new parsers, ...) can be added to Taupe simply by extending the correct interface (or the correct abstract class) with a new class by putting it in the right package. Taupe can dynamically load these classes without further other modifications in Taupe's code. For example, to create a new group of subjects in the system, a developer only need to create a new class that extends the abstract class `Group` and add this new class in the package `laigle.taupe.viewer.utils.group.data`. There are five abstract methods to be implemented in order to make the group effective:

[**getName() : String**] This method gives the name of the group according to its members' characteristics. For example, "Gender" would be the name for a group whose members are organised according to their gender.

[**getPrefixName() : String**] This methods returns the short name of the group and is used as an alternative when, for example, outputting groups in a CSV file.

[**getAvailableValues() : Iterator<Object>**] This method returns the possible values of the characteristics of the group members. Typically, the `GenderGroup` would have three available values: *female*, *male*, and *unknown*.

[**newInstance(Object o) : Group**] This method plays the role of constructor with an argument that represents the common characteristic's value of the group members.

	v0.1	v1.0	v2.0
Total Number of Lines	3193	8036	55698
Total Lines of Code	812	4912	12488
Number of Packages	3	11	25
Number of Classes	10	71	150
Number of Interfaces	0	10	7
Number of Methods	40	490	820
Number of Attributes	30	248	338
Lack of Cohesion of Methods	0.329	0.370	0.224

Table 1: Metric values computed on TAUPE source code

[**isEligible(s : SubjectData) : Boolean**] This method checks whether or not the subject `s` can be added to the group according to its characteristics.

Second, the extensibility of TAUPE also depends on the ease of extending it by developers. A complete documentation is available: both a user's guide and a developer's guide are available on-line[8]. The developer's guide includes explanations to extend eight different kinds of core components of TAUPE. It was written with the objective to ensure that all future developers of TAUPE clearly understand how TAUPE works and to describe the means to add new and–or modify existing components. The system source code is extensively documented using the *Javadoc* mechanism, with more than 820 methods of the system described.

Consequently, new components can be easily added to TAUPE by other researchers.

3.5.3. Validation and Verification

The development of the TAUPE system has included explicit validation and verification phases. As the TAUPE system is used in scientific research, it was crucial that all of its results are correct. The validation process [57] for TAUPE led to the development of several test cases using the *JUnit framework*[9]. The package `laigle.taupe.tests` contains all of the current 50 test cases. These test cases use as oracles values computed by hand for the different implemented algorithms. For example, they include test cases for the computation of the convex hull of a set of fixations or the computation of the *edit distance* between two areas of interest, using an oracle built manually.

The verification process [57] of TAUPE also led to the development of test cases that target the inner working of the system. In particular, several test cases target the core components of the system and the implementations of the various design patterns.

Finally, to give an overall idea of the quality of the current version, Table 1 reports several metrics computed using Eclipse's *Metrics* plug-in[10] on the three available versions of the system.

4. Case Studies

We now summarise three case studies that show the use and the relevance of TAUPE to edit, visualise, and analyse eye-tracking data to further our understanding of program comprehension. The first case study was conducted by Jeanmart *et al.* [54] on the impact of the Visitor design pattern on comprehension and modification tasks; the second study was performed by van den Plas *et al.* [59] on the impact of the Composite and Observer design patterns on comprehension and modification tasks; and the third study was realised by De Smet, Lempereur *et al.* on the impact of the MVC architectural style on comprehension and modification tasks. For each case study, we succinctly recall its goal, null hypothesis and alternative hypotheses, design, and results; then we describe the use of TAUPE in the study.

[9] http://www.junit.org/
[10] http://metrics.sourceforge.net/

4.1. Impact of the Visitor Design Pattern

Goal. The goal of this study was to analyse the impact of the Visitor design pattern [17] on comprehension and modification tasks in the context of the maintenance of programs.

Hypotheses. The study hypotheses were:

- HC_{0_1}: A class diagram with the Visitor does not reduce the subjects' efforts during program comprehension when compared to a class diagram without it.
- HC_{0_2}: A class diagram using the canonical representation of the Visitor does not reduce the subjects' efforts during program comprehension when compared to a class diagram using the Visitor with another layout.
- HM_{0_1}: A class diagram with the Visitor does not reduce the subjects' efforts during program modification when compared to a class diagram without.
- HM_{0_2}: A class diagram using the canonical representation of the Visitor does not reduce the subjects' efforts during program modification when compared to a class diagram using the Visitor with another layout.

If the previous null hypotheses were rejected, it would be possible to assume that, with respect to the threats to the study validity, the following alternative hypotheses were verified:

- HC_{a_1}: A class diagram with the Visitor reduces the subjects' efforts during program comprehension when compared to one without it.
- HC_{a_2}: A class diagram using the canonical representation of the Visitor reduces the subjects' efforts during program comprehension when compared to one using the Visitor and another layout.
- HM_{a_1}: A class diagram with the Visitor reduces the subjects' efforts during program modification when compared to a class diagram without it.
- HM_{a_2}: A class diagram using the canonical representation of the Visitor reduces the subjects' efforts during program modification when compared to one using the Visitor and another layout.

Design. Three open source projects were used in this experiment: *JHotDraw*[11], *JRefactory*[12] and *PADL*[13]. *JHotDraw* is a framework to implement technical and structured drawings, it provides support for the creation of geometric and user-defined shapes. *JRefactory* is a code refactoring tool for Java programs. *PADL* is a meta-model for describing object-oriented programs, it is similar to the UML meta-model.

The dependent variables chosen in the experiment were the *Average Number of Relevant Fixations* and the *Average Duration of Relevant Fixations* combined in the $NORM\_RATE_i$ measure as described in Section 3.3.3.

The eye-tracking device used in this study was the Eye-link II; 24 subjects participated in the study.

Results. The results of this study were that the presence of the Visitor design pattern as well as its layout do not have a significant impact on the comprehension of UML class diagrams when the subjects must perform comprehension and modification tasks but it has a statistically significant impact on modifications [31].

[11] http://www.jhotdraw.org
[12] http://jrefactory.sourceforge.net/
[13] http://wiki.ptidej.net/doku.php?id=padl

Relevance of TAUPE. Before collecting the eye-tracking data, Jeanmart *et al.* decided to use the $NORM\_RATE$ measure to analyse the data and statistically test the various null hypotheses. Jeanmart first implemented the formula in an Excel sheet, which he planned to use by copying/pasting the eye-tracking data into the sheet. It become quickly painfully apparent that copying/pasting thousands of pieces of data collected for each subject was a daunting and erroneous task. Then, Jeanmart implemented the formula as a command in the TAUPE system. After less than a week of implementation and validation and verification, he was able to analyse all the collected data and statistically test his null hypotheses using the TAUPE user interface. The measure and statistical analyses are now available in TAUPE for future studies.

4.2. Impact of the Composite and Observer Design Patterns

Goal. The goal of this experiment was to analyse the impact of the Composite and Observer design pattern [17] comprehension and modification tasks.

Hypotheses. The null hypotheses of the study were:

- H_{0_1}: The impact of the Composite design pattern on the average effort of subjects to perform comprehension and modification tasks is the same for beginners and for experts.
- H_{0_2}: The impact of the Observer design pattern on the average effort of subjects to perform comprehension and modification tasks is the same for beginners and for experts.

If the previous null hypotheses were rejected, it would be possible to assume that, with respect to the threats to the study validity, the following alternative hypotheses were verified:

- $H_{\alpha_{1.1}}$: the presence of the Composite design pattern decreases the average effort of experts when compared to beginners.
- $H_{\alpha_{1.2}}$: the presence of the Composite design pattern increases the average effort of experts when compared to beginners.
- $H_{\alpha_{2.1}}$: the presence of the Observer design pattern decreases the average effort of experts when compared to beginners.
- $H_{\alpha_{2.2}}$: the presence of the Observer design pattern increases the average effort of experts when compared to beginners.

Design. Three open-source programs were selected to compose the questions of the study: *JUnit*, *QuickUML*[14] and *ArgoUML*[15]. JUnit is a unit test framework for the Java programs. QuickUML is a diagramming program to draw UML class diagrams. ArgoUML is also a diagramming program that supports all UML diagrams and provides reverse-engineering facilities as well as exporting in various format.

Four metrics were selected as independent variables in this study to assess the subjects' effort: the *spatial density*, the *transitional matrix*, the *average fixation's duration*, and the *ON target ALL* measure. Complete descriptions of the implementations of these in TAUPE are available in TAUPE user's guide[8]. The level of expertise of the subjects was assessed using their employment. A subset of the subjects were experts from the Pyxis software company[16] while other subjects were students in the Ptidej Team[7] and the Soccer Laboratory[17].

The eye-tracker used to conduct this study was the Eye-link II; 24 subjects took part in the study.

[14] http://sourceforge.net/projects/quj/
[15] http://argouml.tigris.org/
[16] http://www.pyxis-tech.com/en/home
[17] http://web.soccerlab.polymtl.ca/

	ArgoUML	JUnit	QuickUML
Beginners	75.3	84.5	95.5
Experts	106	151.3	88.9
Experts/Beginners (%)	140.8	179.0	107.4

Table 2: Average edit distance between group of subjects per program and ratios

Results. It was not possible to reject the null hypotheses using the *ON target ALL* measure. However, the analysis of the subjects visual paths shows significant commonalities among experts on the one hand and beginners and the other and significant differences between the experts' visual paths and the beginners'. Table 2 reports that the average edit distance between the beginners' visual paths is always lower than that of experts' visual paths, independently of the considered program. This observation shows that beginners systematically browse a diagram while experts use their expertise to quickly gather the important information from the diagram.

Relevance of TAUPE. To the best of our knowledge, the TAUPE system is the only such system providing analyses of the visual path and able to compute the edit distance among a set of visual paths. Thus, it was instrumental in showing the commonalities among beginners and experts and the differences between beginners and experts.

4.3. Impact of Different Forms of the MVC Design Pattern

Goal. The goal of this experiment was to analyse the impact of different forms of the Model View Controller (MVC) architectural style on maintainability tasks.

Hypotheses. The null hypothesis was:

- H_{0_1}: The different forms of the MVC architectural style are all equivalent during the maintenance of a program. The time and the visual path needed to complete the maintenance tasks on UML diagrams are the same, no matter the form of the MVC.

The associated alternative hypotheses were:

- H_{α_1}: The X form of the MVC architectural style is more efficient than the Y form during the maintenance of the programs. The time and the visual path needed to complete the maintenance tasks on UML diagrams are shorter for the X form compared to the Y form.

Where X and Y are two different forms of the MVC design pattern.

Design. The class diagrams of two different programs were obtained by reverse-engineering and studied: the diagram of the `JTable` from the *Java's Swing GUI* widget toolkit[18] and that of *JFreeChart*[19], an open-source framework for the programming language Java, which allows the creation of complex charts.

Different variations of the MVC were then implemented from these two diagrams and used in the study: the canonical form of the MVC style [17], the *Model-Delegate* style [56] (also called *UI-Model* or *Document-View*, MD), and the *Model View Presenter* style [35] (MVP). These variations are shown in FIGURE 9.

The eye-tracking system used in this study was *FaceLAB* as described in 2.3.2. A total of 23 subjects participated to the study. However, only 18 subjects out of the 23 were valid. The "mortality" was due to the instability of *GazeTracker*, which fails to save the data on the hard drive sometimes, and also to the poor quality of the recorded data at first.

[18] http://java.sun.com/products/jfc/tsc/articles/architecture/
[19] http://www.jfree.org/jfreechart/

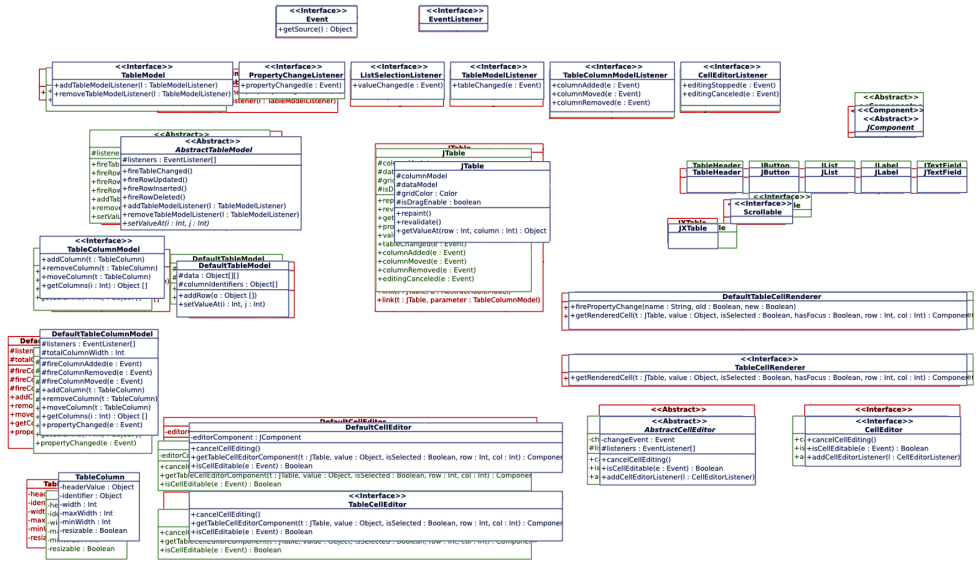


Figure 9: Superposition of all the studied forms of the MVC style

	Comprehension			Modification		
	MD	MVC	MVP	MD	MVC	MVP
Average time (sec.)	66	87	47	40	64	41
Correctness (%)	54	77	60	54	77	60

Table 3: Results collected for the different forms of the MVC architectural style

Results. Table 3 reports the main data collected during this study. The analysis of this data showed that subjects spent less time on the MVP style and that their answer were less correct with the MD style. Average times showed that the MVP style is usually easier to understand than the two other styles. The correctness of the subjects' answers showed that the MVC style is easier to understand than the others forms.

Relevance of Taupe. Again, Taupe eased the process of analysing the recorded eye-tracking data by allowing us to integrate the algorithms necessary to compute the various measures as commands of the system. Moreover, we also used its visual path algorithm to further analyse the collected data, which results we plan to release in the next future.

5. Summary

We now summarise our contribution and future improvements.

5.1. Contributions

To the best of our knowledge, Taupe is the first open-source analysis system for eye-tracking data built with neutrality and extensibility as main objectives. Taupe is released under the GPL licence. Neutrality is realised through the implementations of parsers for the data collected by two eye-tracking devices and the ability to add new such parsers. Extensibility is achieved by providing clear sets of application programming interfaces (APIs) to the various core components of the system and by using reflection to load automatically new implementations.

Taupe is designed to be able to accept multiple eye-tracking systems in input because eye-tracking techniques evolves every day and some new devices appear on the market. The fact that Taupe is easy to evolve can contribute to the development of the field of eye-tracking.

Taupe offers a set of algorithms of analyses in the field of eye-tracking. Although some of these algorithms were well-known and widely used in the field, others are offered for the first time in such a system, for example the analyses of the visual paths and their edit distances.

5.2. Improvements

In the next versions of Taupe, we plan to add a feature to manually correct fixations with some constraints based on their relative distances for example. These constraints could be used when there is no static offset but an offset that is different in different areas of interest.

We also plan to extend the *AOIMaker* so that it can be used to manipulate fixations and saccades recorded by an eye-tracking device, for example to offset the data through drag and drop of the mouse to ease the users' analysis tasks.

We will also improve the performances of the Taupe system in general and of its parsers in particular, which can be time consuming when thousands of fixations and saccades have been recorded, typically during long experimental sessions.

The usability of the Taupe system could be improved by the introduction of graphical user interfaces allowing the users to interact directly with all of Taupe input files, for example to input some information about the subjects instead of editing manually the `.subject` files.

The Taupe system feedback to the users could also be improved by showing more information though the progress bar and the list in the main window. A more thorough usage of the progress bar would improve the user interface. Data and results could also be displayed using charts to improve their visual analyses. Merging the fixations of a subject and colorising them according to their durations would allow Taupe to generate heatmaps [60], which could further help researchers in identifying relevant area of interests.

Obviously, users can use the files generated with Taupe in some external data-mining software systems, such as R[20]. Therefore, we also will create a specific *printer* for such systems. We will also study the feasibility of generating directly `.xls` or `.ods` files.

The current version of Taupe only uses fixations and saccades collected by the eye-tracker devices but new devices provide new kind of data, such as pupil size, head position, blinking rate and so on. Future versions should use this data to allow the development of even more sophisticated measures and analyses.

6. Conclusion

The activity of program comprehension has been studied by many researchers but only recently visual data have be gathered using eye-tracking devices to further improve our understanding of program comprehension processes. An eye-tracker records the coordinates of a subject's gaze when looking at a computer screen. It provides a new perspective on a subject's comprehension process because it shows the areas attracting the subject's attention as well as the visual path of her gaze on the screen [14]. The subject's attention and visual path together form a window on her cognitive processes [45]. Thus, analysing the data recorded using an eye-tracking device allows understanding in details a subject's process of acquiring data, for example during program comprehension.

However, there were still some major obstacles to the widespread adoption of eye-tracking devices. Besides the costs of such devices, the provided analysis software systems were not open-source and often not extensible, preventing the development and seamless integration of new sophisticated analyses [29].

Consequently, we undertook the development of the Taupe system to visualise, analyse and edit eye-tracking data with the main objectives of neutrality and extensibility. We presented the Taupe system: its context, implementation (based on good practices and a thorough documentation, validation, and verification process), and three case studies using Taupe.

[20] http://www.r-project.org/

TAUPE is being developed by the *Ptitdej Team* and released under the GPL and its development will continue in the future to improve its architecture, design, user interface, and to provide more sophisticated measures and analyses. We encourage researchers and practitioners to download its source code[8] and to contribute with measures and analyses of their own.

References

[1] , . Seeing machine's website - facelab. http://www.seeingmachines.com/product/facelab/. Accessed December 23 in 2010.

[2] , 2006. EyeLink II User Manual version (07/02/2006). SR Research Ltd.

[3] , 2009. GazeTracker Reference Manual. Eye Response Technologies Inc.

[4] Aho, A.V., Hopcroft, J.E., Ullman, J.D., 1974. The Design and Analysis of Computer Algorithms. Addison-Wesley. 1^{st} edition.

[5] Bednarik, R., Tukiainen, M., 2006. An eye-tracking methodology for characterizing program comprehension processes, in: Proceedings of 5^{th} symposium on Eye Tracking Research & Applications, ACM Press. pp. 125–132.

[6] Bellay, B., Gall, H., 1997. A comparison of four reverse engineering tools, in: Baxter, I., Quilici, A. (Eds.), Proceedings of the 4^{th} Working Conference on Reverse Engineering, IEEE Computer Society Press. pp. 2–11.

[7] Binkley, D., Davis, M., Lawrie, D., Morrell, C., 2009. To camelcase or under_score, in: Proceedings of the 17^{th} International Conference on Program Comprehension, pp. 158–167.

[8] Boehm, B., Rombach, H.D., Zelkowitz, M.V., 2005. Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili. Springer-Verlag. 1^{st} edition.

[9] Boehm, B.W., 2002. Software engineering economics. Springer-Verlag New York, Inc., New York, NY, USA. pp. 641–686.

[10] Brooks, R., 1978a. Using a behavioral theory of program comprehension in software engineering. IEEE Press, Piscataway, NJ, USA.

[11] Brooks, R., 1978b. Using a behavioral theory of program comprehension in software engineering, in: Wilkes, M.V., Belady, L., Su, Y.H., Hayman, H., Enslow, P. (Eds.), Proceedings of the 3^{rd} International Conference on Software Engineering, IEEE Computer Society Press. pp. 196–201.

[12] Chabris, C.F., Kosslyn, S.M., 2005. Representational correspondence as a basic principle of diagram design, in: Knowledge and Information Visualization, Springer-Verlag. pp. 36–57.

[13] C.Purchase, H., Allder, J.A., Carrington, D., 2002. Graph layout aesthetics in UML diagrams: User preferences. journal of Graph Algorithms and Applications 6, 255–279.

[14] Duchowski, A.T., 2007. Eye tracking methodology. Theory and Practice , 328.

[15] Eichelberger, H., 2003. Nice class diagrams admit good design?, in: Stasko, J.T. (Ed.), Proceedings of the 1^{st} symposium on Software Visualization, ACM Press. pp. 159–168.

[16] Endres, A., Rombach, D., 2003. A Handbook of Software and Systems Engineering. Addison-Wesley. 1^{st} edition.

[17] Erich Gamma, Richard Helm, R.J., Vlissides, J., 1995. Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley Pub Co.

[18] Fitts, P.M., Jones, R.E., Milton, J.L., 1950. Eye movements of aircraft pilots during instrument-landing approaches. Aeronautical Engineering Review 9, 24–29.

[19] Gamma, E., Beck, K., 1998. Test infected: Programmers love writing tests. Java Report 3, 37–50.

[20] Goldberg, J.H., Kotval, X.P., 1999. Computer interface evaluation using eye movements: methods and constructs. International Journal of Industrial Ergonomics 24, 631–645.

[21] Guéhéneuc, Y.G., 2004. A reverse engineering tool for precise class diagrams, in: Singer, J., Lutfiyya, H. (Eds.), Proceedings of the 14^{th} IBM Centers for Advanced Studies Conference (CASCON), ACM Press. pp. 28–41. 14 pages.

[22] Guéhéneuc, Y.G., 2006. TAUPE: Towards understanding program comprehension, in: Erdogmus, H., Stroulia, E. (Eds.), Proceedings of the 16^{th} IBM Centers for Advanced Studies Conference (CASCON), ACM Press. pp. 1–13. 13 pages.

[23] Guéhéneuc, Y.G., 2009. A theory of program comprehension—joining vision science and program comprehension. International Journal of Software Science and Computational Intelligence (IJSSCI) 1. 47 pages.

[24] Gusfield, D., 1997. Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York, NY, USA.

[25] Hadar, I., Hazzan, O., 2004. On the contribution of UML diagrams to software system comprehension. journal of Object Technology 3, 143–156.

[26] Hamlet, D., 2005. Foundations of Empirical Software Engineering. volume 19. Springer Berlin Heidelberg.

[27] ISO/IEC, 1996. EBNF Grammar Specification. Technical Report ISO/IEC 14977.

[28] Jackson, D., Waingold, A., 1999. Lightweight extraction of object models from bytecode, in: Garlan, D., Kramer, J. (Eds.), Proceedings of the 21^{st} International Conference on Software Engineering, ACM Press. pp. 194–202.

[29] Jacob, R.J.K., Karn, K.S., D, P., 2002. Commentary on section 4. eye tracking in human-computer interaction and usability research: Ready to deliver the promises.

[30] Jeanmart, S., 2008a. Evaluation de l'impact d'un patron de conception sur la comprehension et la maintenance de programmes - une experimentation par un systeme d'eye-tracking. Master's thesis. Facultes Universitaires Notre-Dame de la Paix. Namur, Belgium.

[31] Jeanmart, S., 2008b. A study of the impact of design patterns on program comprehension and maintenance activities. ACM SIGSOFT 2008/FSE 16 .

[32] Jørgensen, M., Sjøberg, D.I., 2004. Generalization and theory-building in software engineering research, in: Linkman, S. (Ed.), Proceedings of the 8^{th} international conference on Empirical Assessment in Software Engineering, IEEE Computer Society Press. pp. 29–36.
[33] Just, M., Carpenter, P.A., 1976. Eye fixations and cognitive processes. Cognitive Psychology 8, 441 – 480.
[34] Lehman, M.M., 1980. Programs life cycles and laws of software evolution. Proceedings of the IEEE 68, 1060–1076.
[35] M., P., 1996. Mvp: Model-view-presenter - the taligent programming model for c++ and java. Taligent, Inc. .
[36] von Mayrhauser, A., 1995. Program comprehension during software maintenance and evolution. IEEE Computer 28, 44–55.
[37] von Mayrhauser, A., Vans, A.M., 1995. Program Comprehension During Software Maintenance and Evolution. IEEE Computer Society Press, Los Alamitos, CA, USA. volume 28. pp. 44–55.
[38] Murphy, G.C., Kersten, M., Robillard, M.P., Čubraniś, D., 2005. The emergent structure of development tasks, in: Black, A.P. (Ed.), Proceedings of the 19^{th} European Conference on Object-Oriented Programming, Springer-Verlag. pp. 33–48.
[39] Newell, A., 1973. You can't play 20 questions with nature and win, in: Chase, W. (Ed.), Visual Information Processing. Academic Press.
[40] Palmer, S.E., 1999. Vision Science: Photons to Phenomenology. The MIT Press. 1^{st} edition.
[41] Perry, D.E., Wolf, A.L., 1992. Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes 17, 40–52.
[42] Poole, A., Ball, L.J., 2004. In search of salience: A response time and eye movement analysis of bookmark recognition. People and Computers XVIII-Design for Life: Proceedings of HCI 2004 .
[43] Poole, A., Ball, L.J., 2006. Eye tracking in human-computer interaction and usability research: Current status and future prospects. Ghaoui, Claude (Ed.). Encyclopedia of Human Computer Interaction .
[44] Rajlich, V., 2002. Program comprehension as a learning process, in: Wang, Y. (Ed.), Proceedings of the 1^{st} International Conference on Cognitive Informatics, IEEE Computer Society Press. pp. 343–347.
[45] Rayner, K., 1998. Eye movements in reading and information processing: 20 years of research. Psychol Bull 124, 372–422.
[46] Sharif, B., Maletic, J.I., 2010a. An eye tracking study on camelcase and under_score identifier styles. International Conference on Program Comprehension 0, 196–205.
[47] Sharif, B., Maletic, J.I., 2010b. An eye tracking study on camelcase and under_score identifier styles, in: Proceedings of the 18^{th} International Conference on Program Comprehension, pp. 196–205.
[48] Simon, F., Steinbrückner, F., Lewerentz, C., 2001. Metrics based refactoring, in: Sousa, P., Ebert, J. (Eds.), Proceedings of the 5^{th} Conference on Software Maintenance and Reengineering, IEEE Computer Society Press. pp. 30–38.
[49] Soloway, E., 1986. Learning to program = learning to construct mechanisms and explanations. Commun. ACM 29, 850–858.
[50] Sommerville, I., 1996. Software Engineering - Fifth Edition. Addison-Wesley Publishing Company, Reading, MA.
[51] Spinellis, D., 2003. Code Reading: The Open Source Perspective. Addison Wesley. 1^{st} edition.
[52] Storey, M.A.D., Fracchia, F.D., Müller, H.A., 1999. Cognitive design elements to support the construction of a mental model during software exploration. Journal of Systems and Software 44, 171–185.
[53] Gerardo Cepeda Porras, Guéhéneuc, Y.G., 2010. An empirical study on the efficiency of different design pattern representations in UML class diagrams. Empirical Software Engineering (EMSE) 15. 27 pages.
[54] Sébastien Jeanmart, Guéhéneuc, Y.G., Sahraoui, H., Habra, N., 2009. Impact of the visitor pattern on program comprehension and maintenance, in: Miller, J., Selby, R. (Eds.), Proceedings of the 3^{rd} International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE Computer Society Press. 10 pages.
[55] Sun, D., Wong, K., 2005. On evaluating the layout of UML class diagrams for program comprehension, in: Cordy, J.R., Gall, H. (Eds.), Proceedings of the 13^{th} International Workshop on Program Comprehension, IEEE Computer Society Press. pp. 317–326.
[56] Swartz, F., 2004. Ui-model structure. http://www.leepoint.net/notes-java/GUI/structure/30presentation-model.html. Accessed October 18 in 2010.
[57] The Institute of Electrical and Eletronics Engineers, 1990. Ieee standard glossary of software engineering terminology. IEEE Standard.
[58] Uwano, H., Nakamura, M., Monden, A., ichi Matsumoto, K., 2006. Analyzing individual performance of source code review using reviewers' eye movement, in: Proceedings of the 2006 symposium on Eye Tracking Research & Applications, pp. 133–140.
[59] Van Den Plas, B., 2009. La theorie "Vision-Comprehension" appliquee aux patrons de conception. Master's thesis. Facultes Universitaires Notre-Dame de la Paix. Namur, Belgium.
[60] Wilkinson, L., 2009. The history of the cluster heat map. American Statistician 63, 179–184.
[61] Yusuf, S., Kagdi, H., Maletic, J.I., 2007a. Assessing the comprehension of UML class diagrams via eye tracking, in: Stroulia, E., Tonella, P. (Eds.), Proceedings of the 15^{th} International Conference on Program Comprehension, IEEE Computer Society Press. pp. 113–122.
[62] Yusuf, S., Kagdi, H., Maletic, J.I., 2007b. Assessing the comprehension of uml class diagrams via eye tracking, in: Proceedings of the 15^{th} International Conference on Program Comprehension, IEEE Computer Society. pp. 113–122.

Annexe C

Taupe - Guide du développeur

Facultés Universitaires Notre Dame De la Paix (FUNDP Namur)
Faculté d'Informatique

École Polytechnique de Montréal
Département de Génie Logiciel

Team Leader: Yann-Gaël Guéhéneuc

TAUPE 2.0 - Developer's Guide

Benoît De Smet
Lorent Lempereur

May 10, 2011

Contents

List of Figures

1 Introduction

This document aims to help developers to understand and modify the TAUPE 's source code.
The first section gives a set of preliminary tips for the developers.
The following section, using class diagrams and package diagrams, explains how the software is designed and which design have been taken.
Section 4 explains how add or modify some features into the software.
Section 5 describes how to compile the whole software.
TAUPE contains a set of test cases, as explained in Section 6. TAUPE follows the terms of a licence mentionned in Section 7.
Section 8 gives some links to contact the team that developed TAUPE .

2 Good to Know

2.1 About this Document

The source of this document (written in LaTeX) are available at `http://www.ptidej.net/research/taupe/`. Please maintain this document while you modify the software.

2.2 System Requirements

Java

The software is written in Java[1] using the *Java SE 1.6*.

Libraries

The following libraries are needed to develop the current source code:

- `jdom 1.1:` Used to parse the XML files from the *Eye-link II* system[2].
- `junit 4.8.3:` Used for the test cases.

2.3 Version

The whole software was created at the *Ecole Polytechnique de Montreal*[3] in the *Ptidej Team*[4].

`0.0` The first version consists of the raw data of the software (fixations and saccades) and a graphical user interface to visualise the data from Eye-link®II. It was written by Yann-Gaël GUÉHÉNEUC. [1]

`1.0` The version of the software written by several students in internship (mainly by Bertrand VAN DEN PLAS) at the *Ptidej Lab*. The version `1.0` allowed the user to apply some algorithms to some data.

[1] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
[2] `http://www.sr-research.com/EL_II.html`
[3] `http://www.polymtl.ca`
[4] `http://www.ptidej.net`

2.0 During Fall 2010, the whole software was rewritten by Benoit De Smet and Lorent Lempereur. As explained below, this last version is able to parse the files generated by different kinds of eye-tracking systems and contains many new features. It is the current version of the software. The development of this new version lasted a couple of months under the supervision of Yann-Gaël Guéhéneuc.

2.4 Javadoc

We ask to the developers who modify the source code to maintain the whole *Javadoc*. A particular syntax is used:

- Most of the methods uses preconditions about the input parameters. The preconditions are expressed by a comment after the parameter's description. For example, if a developer wants to express that a parameter should be strictly positive:

```
1  /**
2  * Setter
3  * @param price the new price; price > 0
4  */
5  void setPrice(int price) {
6    if (price <= 0) {
7      throw new IllegalArgumentException("The␣price␣must␣be␣
           strictly␣positive.");
8    }
9
10   this.price = price;
11 }
```

- Each class must express its author (with the `@author` annotation) and its version (`@since`) as described in Section 2.3.

2.5 Directories

The project directory is currently organised following the structure shown in figure 1:

- `doc` the generated javadoc, some `README` files concerning the usage of the software including this document.
- `lib` all the libraries
- `src` the source files
  - `icons` the set of icons used in the software (images)
  - `laigle` (*Laboratoire de Génie Logiciel Expérimental* or *Experimental Software Engineering Laboratory*) the main package
    - `taupe` the package that contains the application's code
    - `tests` the classes which define the unit tests
- `rsc` some ressources like `edf2xml` or some examples...
  - `tests` the input files for the unit tests

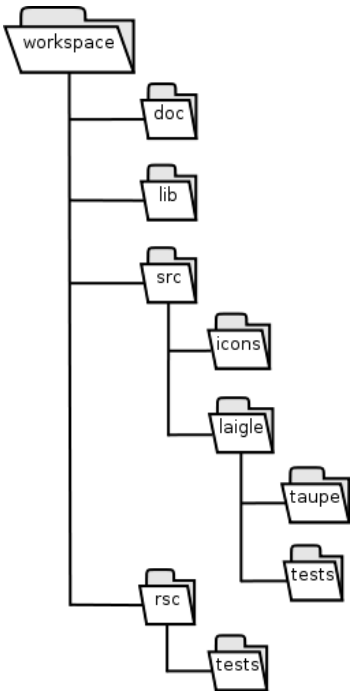


Figure 1: Directories

2.6 Choices

- All the design patterns are based on the definitions from [2].
- As illustrated in Section 2.4, the preconditions are checked using the `IllegalArgumentException` class. Actually, this exception is more often used for the method that have a `public` or `package` visibility according to [3]. For `private` methods, the developers may use an assertion (`assert`).
- The identifier of each variable always uses the Camel Case convention and never the character "_". For example, "max value" is written `maxValue`, not `max_value`.
- The choice of *Java SE 1.6* pertains to the possibily of adding the `@Override` annotation to the methods implemented from interfaces. It also allows the usage of the `Nimbus` look and feel[5].
- Some classes are loaded with the *Java*'s reflection. TAUPE must know the content of some packages to load according to two situations: TAUPE is in development (for example, a developer works in *Eclipse* or *Netbeans* with the `.java` files) or TAUPE is a JAR file. When the software is in development, TAUPE lists some packages using the file system. When TAUPE is a JAR file, it uses the classes `java.util.jar.JarFile` and `java.util.jar.JarEntry` to browse the all JAR file. For this reason, the JAR file cannot be renamed. Altought TAUPE was renamed, the JAR file's name used would be the nearest of `TAUPE.jar` thanks to the *edit distance* algorithm from *Levenshtein*.

[5] https://nimbus.dev.java.net/

3 Program Design

3.1 Packages

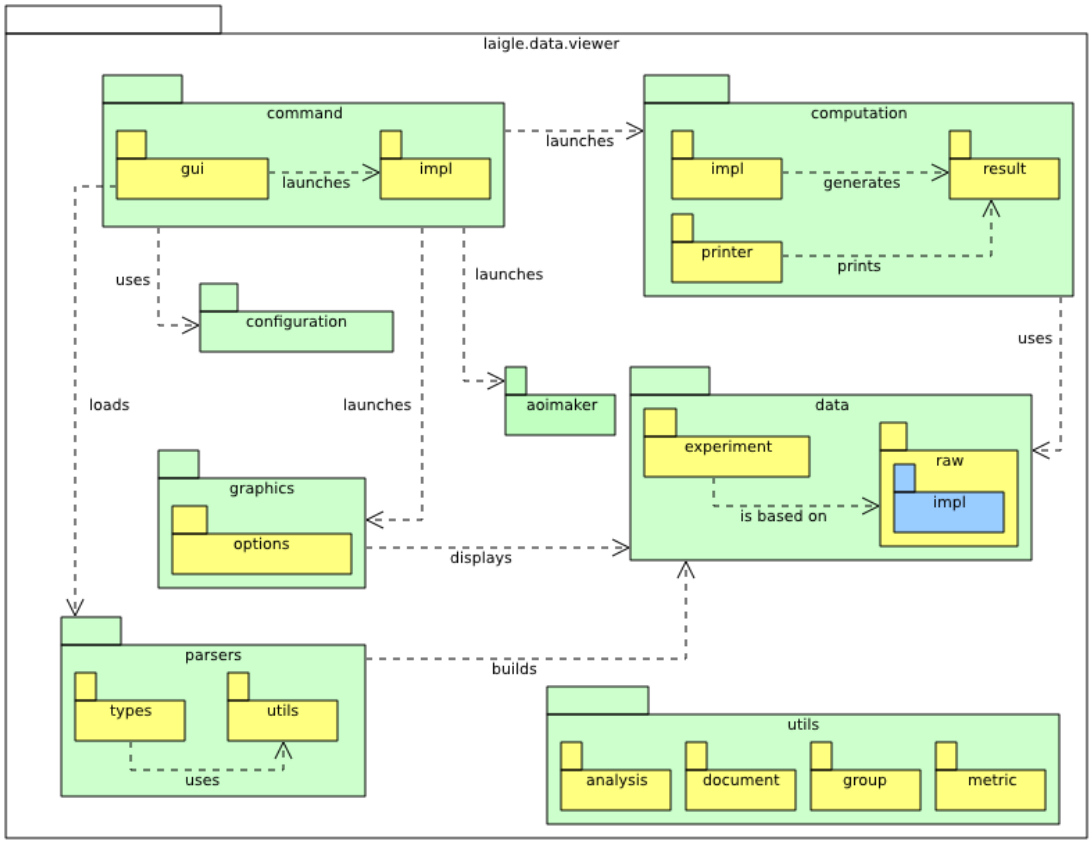


Figure 2: Package diagram

FIGURE 2 shows the project's structure. The next sections describe more precisely the content of the packages.

3.2 Data

This section describes the package `laigle.taupe.viewer.data` (see FIGURE 3).

Description

- **fixation**: A fixation is a position of the eye during a gaze.
- **saccade**: A saccade is a movement of the eye between two fixations.
- **experiment**: The experiment represents the whole system we want to analyse with the software. Actually, it is a set of questions and their answers. When the user is using the software, this one handles up to only one experiment at a time.
- **question**: A question (a `Question`'s instance) is related to one image which defines the whole question.
- **subject**: A subject (a `SubjectData`'s instance) is a person which answered to a set of questions during the time of the experiment. They are defined by a set of characteristics like their name or their level of study. Each subject is linked to a file which is generally generated by an eye-tracking system and that contains the whole set of data about the subject's answers. If the file is called `Subject01.xml`, then it maybe exists a file named `Subject01.subject` which describes the subject's characteristics and whether their answers are wrong or not.
- **area of interest**: An area of interest (a `AreaOfInterest`'s instance) is an area on a question's image. This area can be relevant or not. It also can be "ignorable", it means that the system will not take account of this area. Those areas are defined in a text file for each question. For example, if a question is related to an image called `foo.png`, then the file which defines the areas is named `foo.aoi`.
- **answer**: An answer is what a specified subject has answered to a specified question. This answer can be correct or not (determined by the experiment's supervisor) and is related to a set of fixations and a set of saccades that the subject did during the time of the experiment. All the answers for a subject are in the same file.
- **scene**: This concept represents the link between a question, its areas of interest, and the fixations of each answers for this question.

Choices

The software handles one and only one experiment at a time, therefore the `Experiment`'s instance follows the *singleton* pattern.
The *observer* design pattern is used with the `Experiment`'s singleton to notify the whole software about this singleton's changes.
The software uses an `Integer` to represents the relevance of an area of interest because we assumed that other developers would be interested by using a `weight` with the concept of areas of interest.
The `Experiment`'s attribute named `minDuration` is used to define the minimal duration (in milliseconds) of a fixation. Some eye-tracking systems (like *GazeTracker*^{TM}, for example) sometimes allow to choose this duration. So, TAUPE is able to set this duration too. Therefore, it is easier for the user to make different analysis based on a same set of files. Developers manipulates fixations and saccades with the methods declared respectively in `IFixation` and `ISaccade`.

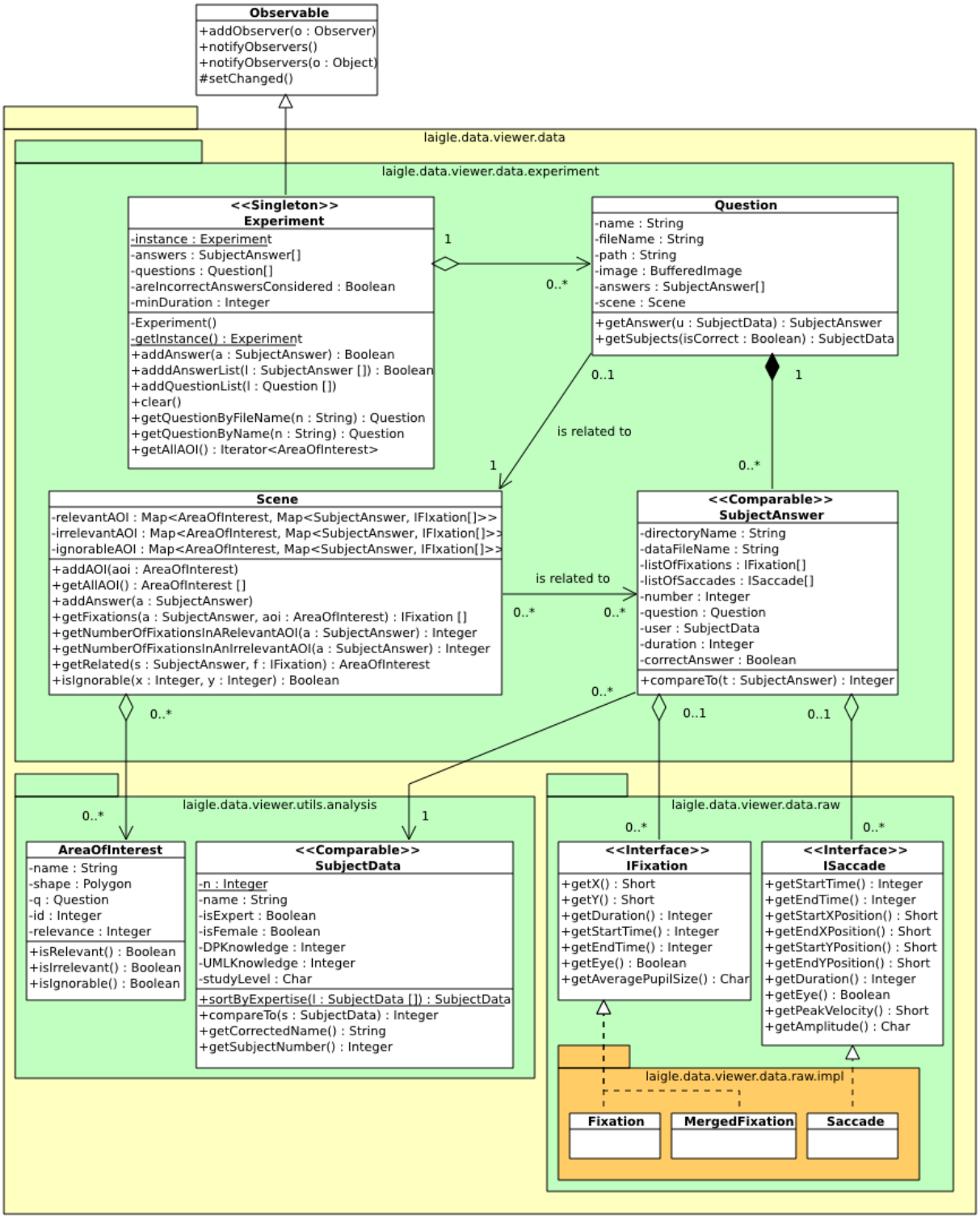


Figure 3: Class diagram - Package `laigle.taupe.viewer.data`

3.3 The Commands

This section describes the package `laigle.taupe.viewer.command` shown in FIGURE 4. A "command" is a significant module in the software. It allows to launch an important module from the graphical user interface. Each command has a set of parameters defined by the class named `Argument`. The user must often set some parameters before the command's execution. The progress bar of the main graphical interface is modified by the progression of those commands and all the notifications in the "information panel" are thrown by their errors or notices. For example, a command can be the feature "Get Cache Summary" or "Graphical Visualisation".

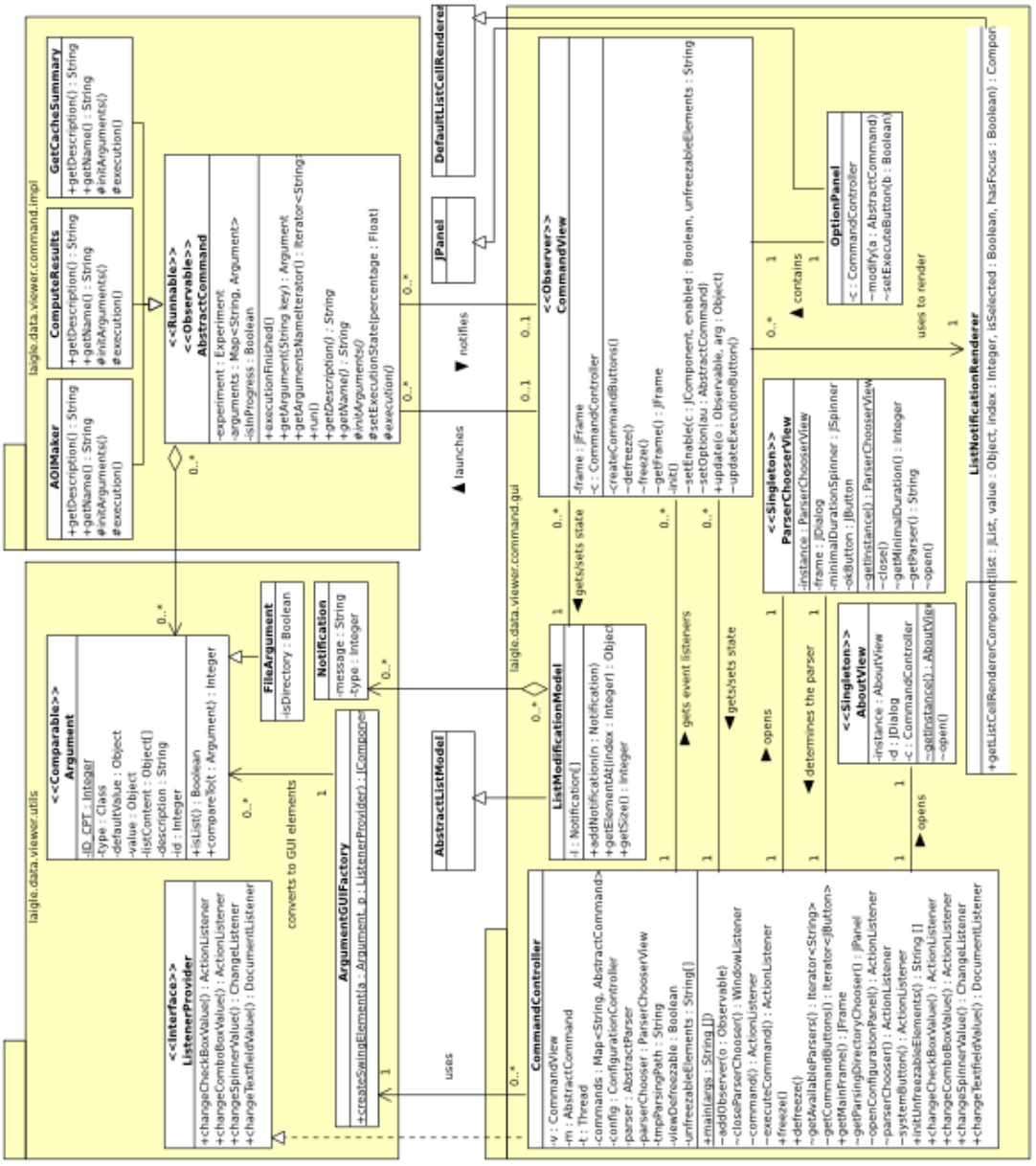


Figure 4: Class diagram - Package `laigle.taupe.viewer.command`

Description

- **command**: The commands are coded through the abstract class `AbstractCommand` and its sub-classes. The reason why the class extends the *Runnable* interface is that it must be executed in parallel with the graphical user interface in a separate thread. Regarding the `Observer` interface, the notifications are thrown to the GUI through the *observer* design pattern.
- **parameter**: An instance of `Argument` can be a simple value or a set of values. If a parameter is an instance of `FileArgument`, it represents a file or a directory in the file system. To help the developer, the class `ArgumentGUIFactory` is able to convert a parameter into a graphical element thanks to the Swing API. The parameters are displayed by the panel coded in the class called `OptionPanel`.
- **notification**: The notifications in the list are described by the class `Notification`, stored in the instances of `ListModificationModel` and displayed through the class `ListNotificationRenderer`. These classes redefine the default Swing classes.
-

Choices

TAUPE , the set of *commands*, and the graphical user interface (GUI) are designed following the classical design pattern *Model-View-Controller* (MVC).
In the package `laigle.taupe.viewer.command`, the main model is represented by the class `AbstractCommand`, the view by the class `CommandView` and the controller by the class `CommandController` that describes the whole set of event listeners related to the main frame of the software.
The choice of the *observer* design pattern for the notifications is justified by the ease to send a notifications to the main GUI.
The interface named `ListenerProvider` is used to define the event listeners that describe how the model is modified by the Swing elements of the `Argument`'s instances.
Some user interfaces are *singleton* to avoid multiple instantiation of the unmodifiable user interface. For example, the "About View" displays every time the same elements.
The process to modify the set of parameters is described in Section 4.1. The process to add a new *command* is described in Section 4.2.

3.4 Preferences

This section describes the package `laigle.taupe.viewer.configuration` (see FIGURE 5). The "Preference" system allows the developer to define a default configuration for TAUPE . Moreover, it allows the user to use this default configuration and to modify its values. The configuration's data displayed through the software are stored in the software's cache memory and are recorded into a `.properties` file.

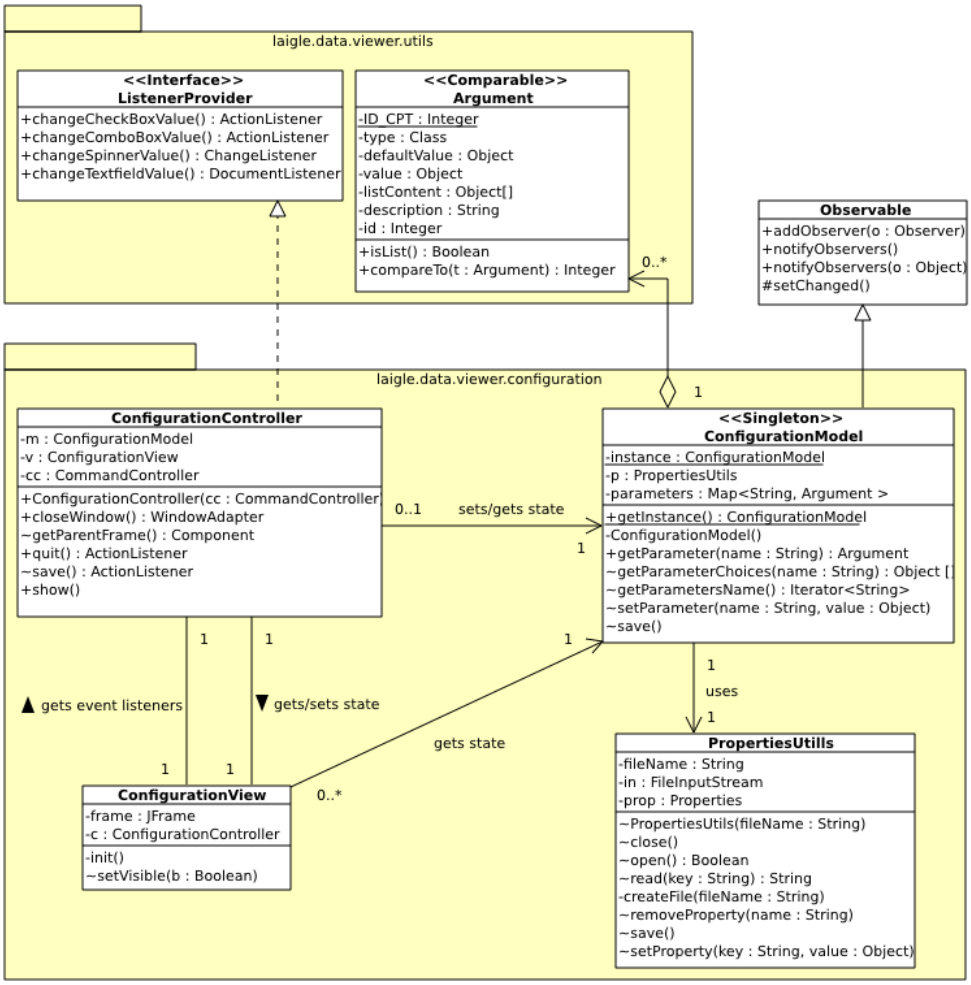


Figure 5: Class diagram - Package `laigle.taupe.viewer.configuration`

Description

The class named `PropertiesUtils` is used to record the preferences permanently *i.e.*, in a `.properties` file. The classes `ListenerProvider` and `Argument` are already defined in Section 3.3.

Choices

The GUI related to the "Preference" system is designed with the MVC where the model is stored in the `ConfigurationModel`'s instance, the view is displayed thanks to the `ConfigurationView` class and the controller, which handles the events about the modification of the configuration by

the user, is defined by the `ConfigurationController`'s instance.
To allow the package `laigle.taupe.viewer.command` to use easily the configuration in the commands' parameters and the parser's chooser, the class `ConfigurationModel` is designed with the singleton design pattern.
The process to modify the set of preferences is described in Section 4.7.

3.5 Groups

This section describes the package `laigle.taupe.viewer.utils.group` (see FIGURE 7). The notion of group allows TAUPE 's users to classify subjects in some (non-exclusive) subsets of subjects in accordance with their characteristics.

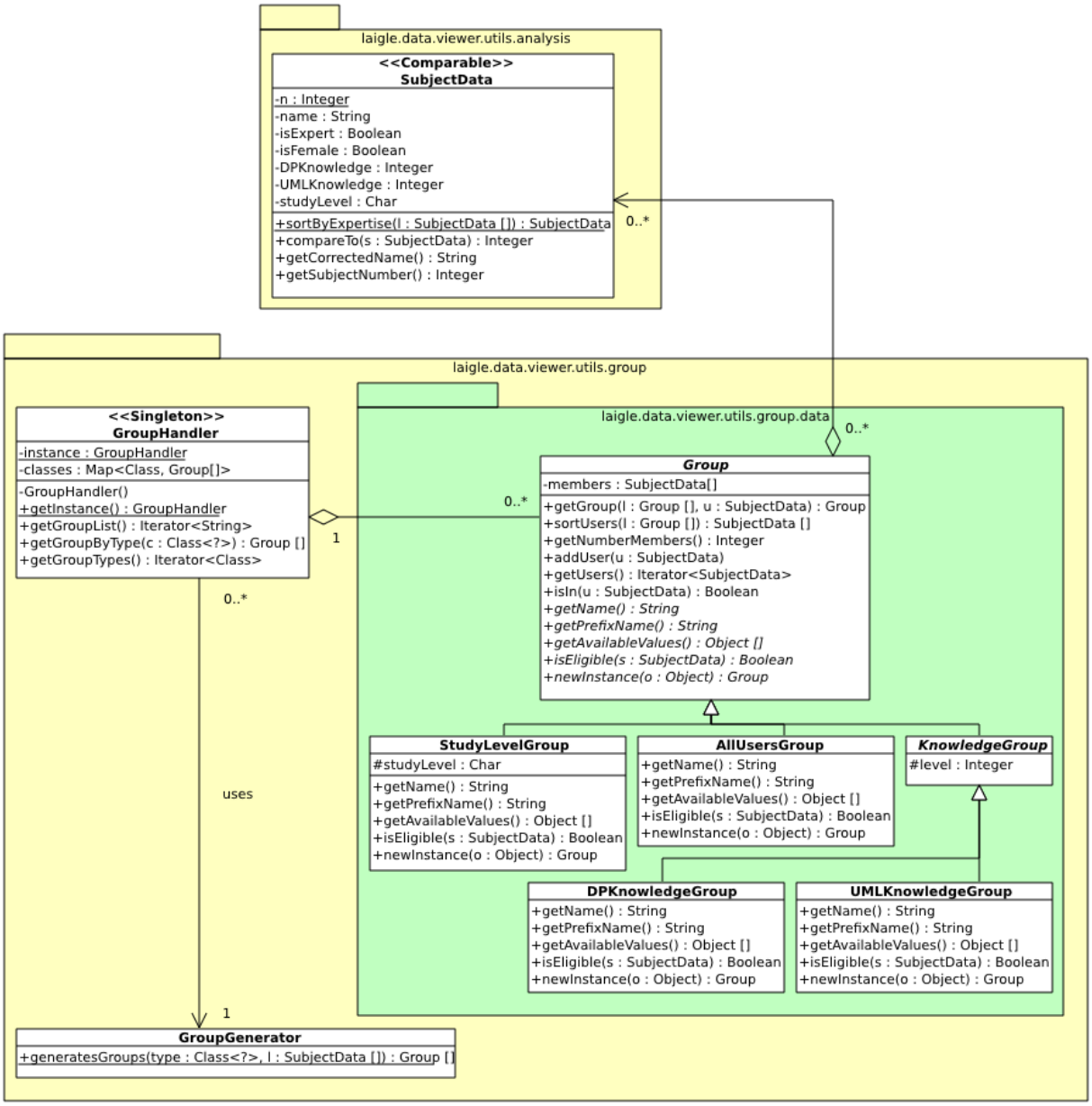


Figure 6: Class diagram - Package `laigle.taupe.viewer.utils.group`

Description

- **group**: A group of subjects. A group is defined by its type and characteristics. For example, groups that classify the subjects based on their UML knowledge are defined by the class

`UMLKnowledge` and inner attribute `level`, which contains the level of its members.

- **subject**: Already defined in Section 3.2

Choices

The kinds of groups are defined by all the sub-classes of `Group`. Those classes are loaded by the Java's system called *reflection*. The *singleton* represented by the class `GroupHandler` is used to handle the whole set of groups. The *singleton*'s instantiation loads the whole set of groups and subjects are added inside each one in a convenient way using *GroupGenerator*.
The process to add a new kind of group is described in Section 4.4.

3.6 Computation

This section describes the package `laigle.taupe.viewer.computation` (see FIGURE 7). This package represents one *command* of the software and is called using the class named `ComputeResults`.

Description

- **result**: The data produced by the classes which implement the interface named `ICompute`, here called `computation`. All the results extend the abstract class called `AbstractResult`.
- **printer**: The tool that generates an output from a result. All the printers implement the interface called .

Choices

The hierarchy of results is represented using the *composite* design pattern. The class `ContainerResult` plays the role of *Composite* and the class `AbstractResult` is the *Component*.
The printers generate some output from the results. To isolate the role of the printing and the role of the browsing, the module uses the *visitor* design pattern. The result hierarchy plays the role of *Element*, it describes, through the method `accept`, how to browse and handle the hierarchy. For example, the method `accept` defined in `ContainerResult` creates a new directory (through the related `visitor` method of the *printer*) and launches `accept` from each its children into the new directory. Therefore, the hierarchy browsing is currently a *depth first search*. The printers play the role of *visitor*, they define a set of methods called *visitXXX* that describe how to print a specified result. They are called by the Java's reflection system.
The method to generate the results is `compute(GroupHandler gh, Experiment e)` from the `ICompute`'s sub-classes. The `GroupHandler`'s singleton is used to sort and classify the results and the `Experiment`'s singleton contains all the data needed by a computation.
This module is launched from the *command* represented by the class `laigle.taupe.viewer.command.impl.ComputeResult`, which allows TAUPE 's users to choose the desired computations and the desired printers. The way to add a new *computation* is described in Section 4.3. The way to add a new *printer* is described in Section 4.6.

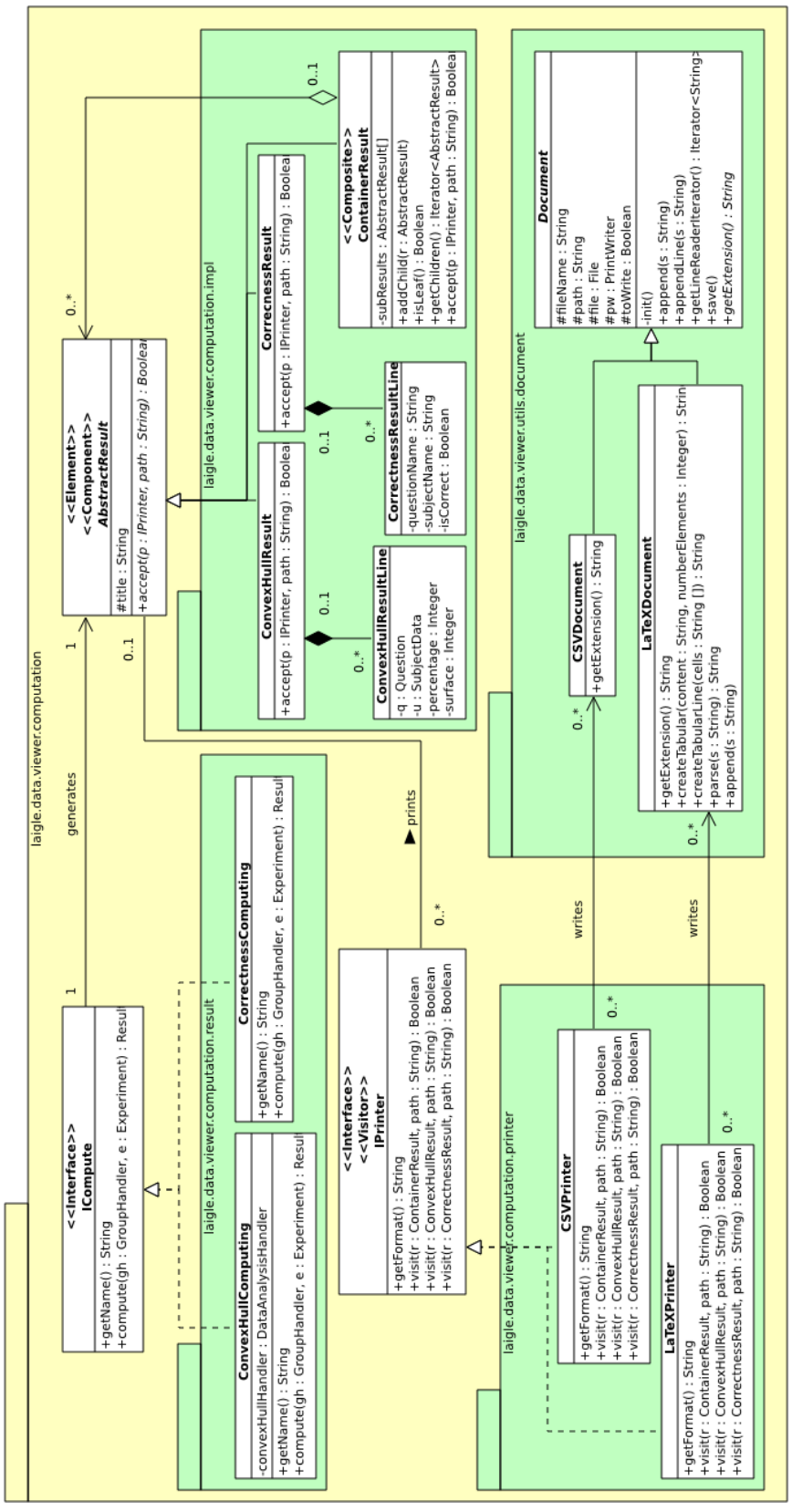


Figure 7: Class diagram - Package `laigle.taupe.viewer.computation`

3.7 Parsers

This section describes the package `laigle.taupe.viewer.parsers` for the eye-tracker's files (see FIGURE 8).

Description

- **parser**: A tool which is used to read some files generated from eye-tracking system and to fill the cache data about the related experiment. All the parsers extend the class named `AbstractParser` and must be member of the package `laigle.taupe.viewer.parsers.types`.

Choices

The parsers implement the interface `Runnable` to be executed in parallel with the GUI. They follow the *observer* design pattern to easily notify the GUI.
The list of parsers in `laigle.taupe.viewer.parsers.types` are loaded using the Java's reflection system and with the class called `Package`. The package `laigle.taupe.viewer.parsers.utils` contains a set of tools to parse some kind of files:

- `AbstractParser` uses a `RelevantSlide`'s instance to know whether or not a file must be taken in account.
- `AbstractParser` uses a `SubjectParser`'s instance to get some information in the `.subject` files.
- `AbstractParser` uses an `AOIParser`'s instance to get the areas of interest about the questions.
- `AbstractParser` uses an `OffsetParser`'s instance to load the information about a hypothetical offset on the questions's image.
- `GazeTrackerParser` uses an instance of a `ILineParser`'s child to parse different kinds of line.

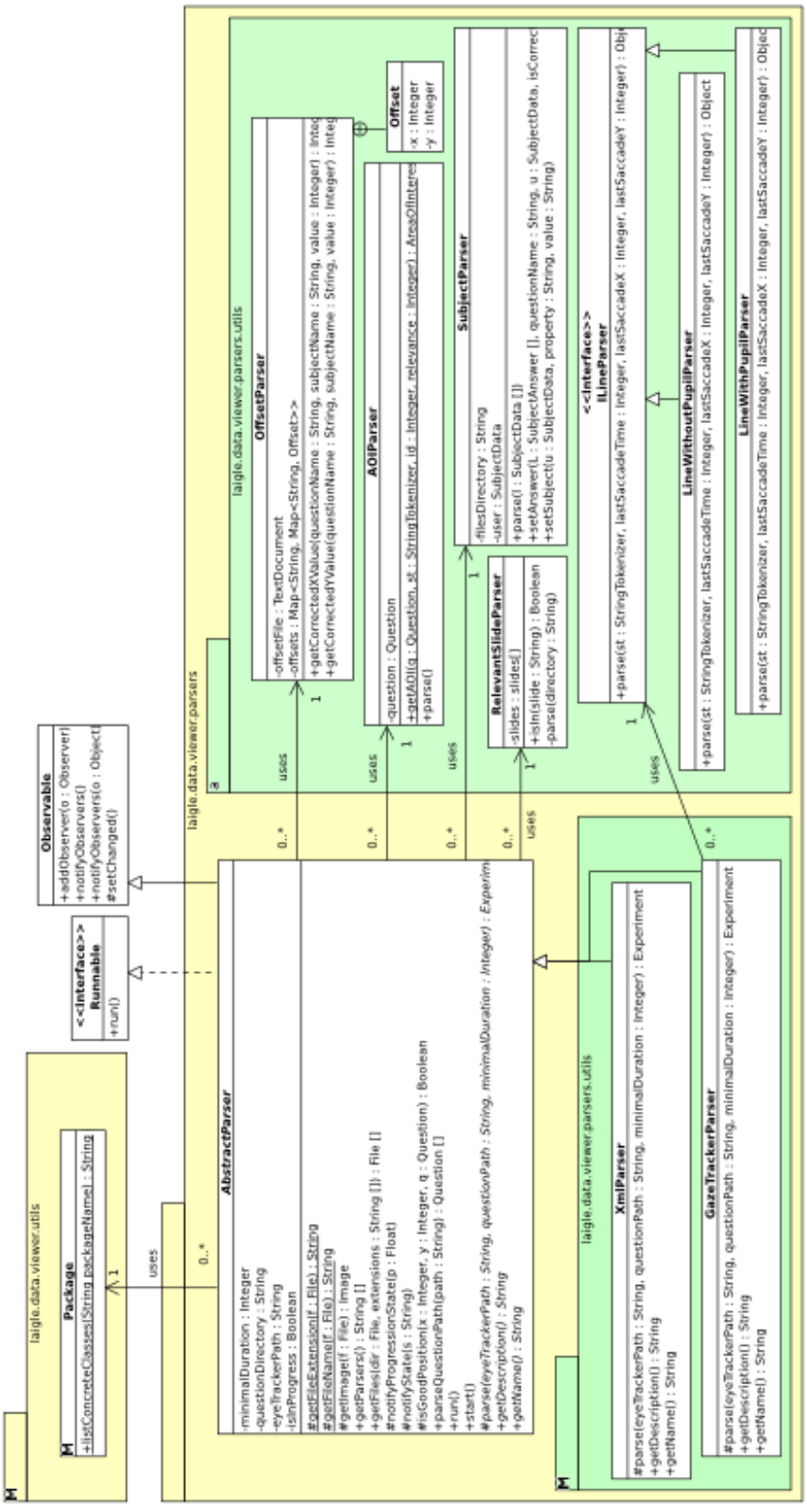


Figure 8: Class diagram - Package `laigle.taupe.viewer.parsers`

3.8 The graphical visualisation system

This section describes the package `laigle.taupe.viewer.graphics` (FIGURE 9). Actually, the graphical visualisation is a *command* of the program. It allow the user to visualise the whole experiment.

Description

- **drawer**: An option is the *visual system* which is able to draw something on the GUI's panel. For example, a drawer is able to draw the images associated with questions.
  All the drawers extend the class named `AbstractDrawer` and are members of the package `laigle.taupe.viewer.graphics.options`.

Choices

This feature follows the *MVC* design pattern where `ExperimentView` is the view, `ExperimentDomain` is the model (more precisely the model domain, as described in [4]), and `ExperimentController` is the controller that defines all the event listeners.
`QuestionView` represents the main panel in the GUI where is drawn all the `AbstractDrawer`'s executions.
The "graphical visualisation" system is launched by the instantiation of the controller.
`ExperimentDomain` implements the interface `Runnable` to notify the progression of the drawing to the `QuestionView`'s instance using the *observer* design pattern.

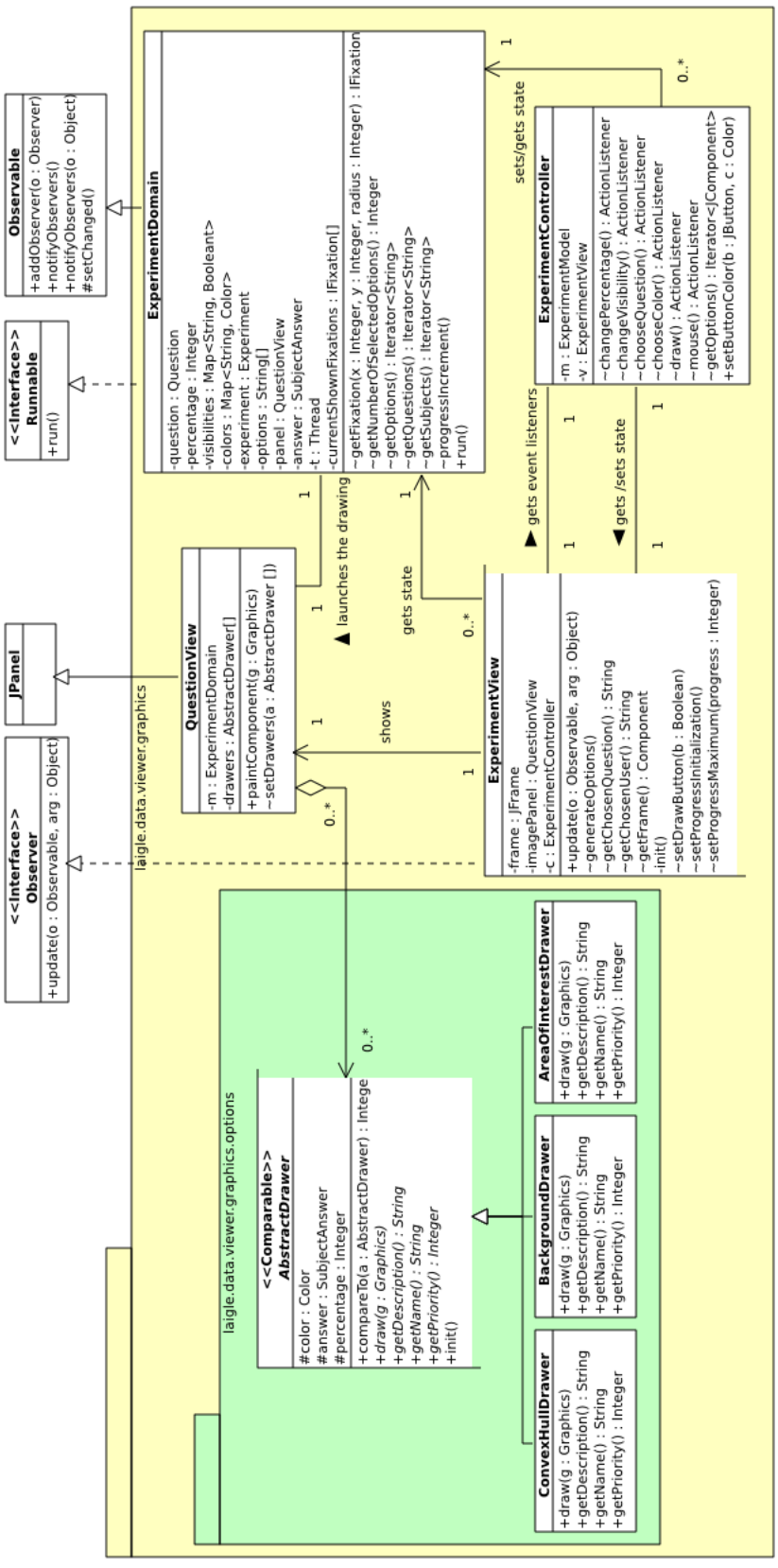


Figure 9: Class diagram - Package `laigle.taupe.viewer.graphics`

3.9 The "AOI Maker" module

This section describes the package `laigle.taupe.viewer.aoimaker` (see FIGURE 10). This module is a *command* of the program. It is used to create the files that contains the description of the areas of interest using a graphical interface.

Choices

This module contains a `public static void main(String[] args)` method that allows it to be launched without launching the whole software. Therefore, the developer may start the module by this `main` method or by the `AOIMakerController` instantiation.
The "AOI Maker" follows the design pattern *Model-View-Controller* (MVC) where the class named `AOIMakerController` is the controller, `AOIMakerModel` is the model, and the main view is `AOIMakerView`. `AOIMakerView` shows a set of panels represented by the `BottomPanel`'s sub-classes. These sub-classes' displays are exclusive, so the methods `showXXX()` defined in `AOIMakerView` are used to show one *bottom panel* at a time. `AOIComboBoxModel` is a part of the model and redefines the default Swing system.
The *observer* design pattern is used to make easier the communication between the model and the view.
The class named `AOIMakerModel` implements `Runnable` to write the result file in parallel with the GUI.

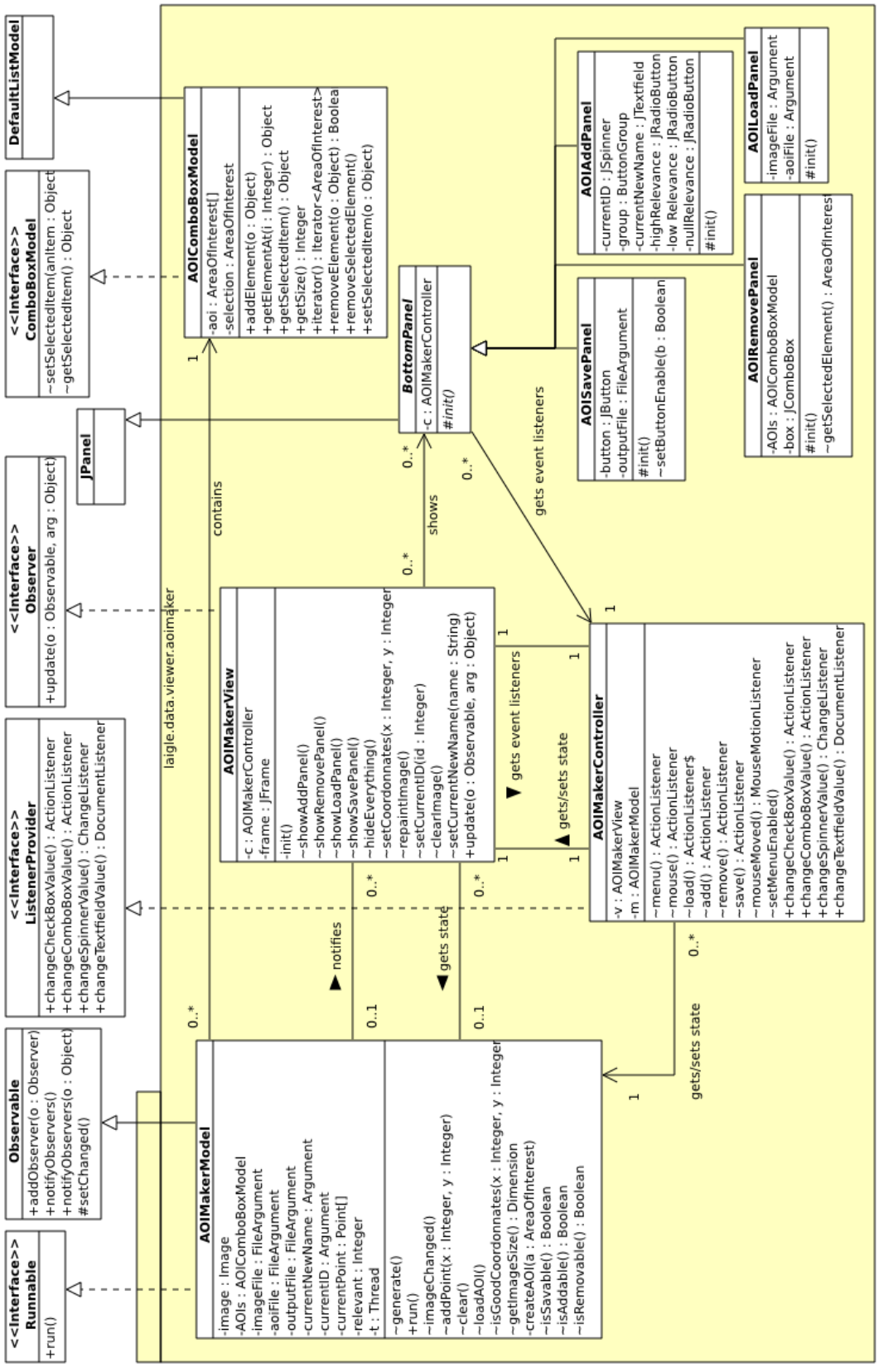


Figure 10: Class diagram - Package `laigle.taupe.viewer.aoimaker`

4 Maintainability

This section explains how developers can add or modify some features of the software. The following list of possible modifications is not comprehensive but is a good start to understand the software following recurring scenarios.

4.1 How to Modify a Set of Parameters

As mentionned in Section 3.3, each *command* has a set of parameters that allows TAUPE 's users to give some arguments to the command's execution.
To add some parameters to a command, a developer must modify the *protected* method named `initArguments()` in the related command's class (which must extend the class named `AbstractCommand`). This method just add a set of instances of `Argument` to the attribute called `arguments`. These arguments will be automatically handled by the graphical user interface (GUI). Then, the developer can use them in the method `run()` of the desired *command*.

Advice

- Declare the name of the new argument as a *constant* in the related command's class. It will make the new argument easier to access.

4.2 How to Add a New *command*

If a developer wants to add a new feature to the software, she must add a class that extend `AbstractCommand` and put it in the package `laigle.taupe.viewer.command.impl`. There are four abstract methods to implement:

[**String getName()**] This method gives the name of the new feature. The name is displayed in a `JButton` in the main frame.

[**String getDescription()**] This method gives the description of the new feature. The description is displayed in a `JLabel` in the panel which contains the arguments.

[**void initArguments()**] The developer must implement this method as described in Section 4.1.

[**void execution()**] This method is the actual content of the feature that will be executed in a new *thread*. This method must notify the software about its progression using the method `notifyView`.

The GUI will automatically take into account of a new *command*. To add a feature which is not represented by a freezed (disabled) button at the software's start, the developer must modify the method `CommandController.initUnfreezableElements` to add the name of the command's class to the attribute `unfreezableElements`.

4.3 How to Add a New Computation

The *computation* package is described in Section 3.6. There are three steps to add a new algorithm. Let us assume that the developer wants to add the addition as an algorithm to the software.

Step 1 Implement the result generated by this new algorithm. A result is a class which extends the class `AbstractResult`. An instance of this new class must contains all the result data.

The developer can use the class `ContainerResult` to describe its result as a hierarchy. All the sub-classes of `AbstractResult` implement the method `accept`:

boolean accept(p : IPrinter, path : String) This method describes how to browse the related result with the printer `p`. For instance, `ConvexHullResult.accept` just calls the `visit` method from the printer but `ContainerResult.accept` calls `visit` on each of its sub-results (*depth first search*).

```
1  package laigle.taupe.viewer.computation.result;
2
3  public class AdditionResult implements AbstractResult {
4
5    private int result;
6
7    @Override
8    public boolean accept(IPrinter p, String path) {
9
10     ....
11
12     return p.visit(this, path);
13   }
14 }
```

Step 2 Create a new class that implements the interface `ICompute` and put it in the package `laigle.taupe.viewer.computation.impl`. There are two methods to implement:

[**getName() : String**] This method gives the name of the new algorithm. This name is displayed in the `JComboBox` in the feature named `ComputeResults`.

[**compute(gh : GroupHandler, e : Experiment) : AbstractResult**] The actual algorithm to be called on an *experiment* (defined in Section 3.2). The developer can use a `GroupHandler` to sort the sub-results.

In this case:

```
1  package laigle.taupe.viewer.computation.impl;
2
3  public class AdditionComputation implements ICompute {
4
5    @Override
6    public String getName() {
7      return "Addition";
8    }
9
10   @Override
11   public AbstractResult compute(GroupHandler gh,
       Experiment e) {
12
13     AdditionResult a = new AdditionResult(...);
14     ...
```

```
15      return a;
16    }
17  }
```

Step 3 Create the printing method for each *printer* for this new result. In this case:

```
1  package laigle.taupe.viewer.computation;
2
3  public interface IPrinter {
4
5    ...
6
7    @Override
8    public boolean visit(final AdditionResult r, final
         String path);
9
10   ...
11
12 }
```

Advices

- Do not use the configuration system (the `ConfigurationModel`'s singleton). Instead, add some parameters to the `ComputeResults` class.
- The order of those steps is not mandatory but it is strongly suggested to create the result first because the two other steps depend on the result's structure.

4.4 How to Add a New Group of Subjects

The notion of group is defined in Section 3.5. To add a new group in the system, the developer just has to create a new class which extends the abstract class named `Group`. She must add this new class in the package `laigle.taupe.viewer.utils.group.data`. There are five abstract methods to implement to make the group effective:

[**String getName()**] This method gives the name of the group according to its members' characteristics.

[**String** getPrefixName()]] A short version of the method `getName()`'s result.

[**Iterator<Object> getAvailableValues()**] The possible values of the characteristic of the group's members.

[**Object newInstance(Object o)**] This method plays the role of constructor with an argument.

[**Boolean isEligible(s : SubjectData)**] This method checks whether or not the subject `s` can be added to the group according to its characteristics.

Let us assume that a developer wants to create three groups related to the gender of their members, she will implement a new class named `GenderGroup`:

```
1  package package laigle.taupe.viewer.utils.group.data;
2
3  public class GenderGroup extends Group {
4
5    protected char gender;
6
7    public static final char MALE = 'M';
8    public static final char FEMALE = 'F';
9    public static final char UNKNOWN = 'U';
10
11   public GenderGroup() {
12     super();
13     this.gender = GenderGroup.UNKNOWN;
14   }
15
16   public GenderGroup(char gender) {
17     super();
18     this.gender = gender;
19   }
20
21   @Override
22   public String getName() {
23     String result = "";
24
25     switch (this.gender) {
26
27       case GenderGroup.FEMALE :
28       result = "Female";
29       break;
30
31       case GenderGroup.MALE :
32       result = "Male";
33       break;
34
35       case GenderGroup.UNKNOWN :
36       result = "Unknown";
37       break;
38
39     }
40     return result;
41   }
42
43   @Override
44   public String getPrefixName() {
45     String result = "";
46
47     switch (this.gender) {
```

```
48
49          case GenderGroup.FEMALE :
50          result = "F";
51          break;
52
53          case GenderGroup.MALE :
54          result = "M";
55          break;
56
57          case GenderGroup.UNKNOWN :
58          result = "U";
59          break;
60        }
61        return result;
62      }
63
64      @Override
65      public Iterator<Object> getAvailableValues() {
66        final List<Object> l = new ArrayList<Object>();
67        l.add(GenderGroup.MALE);
68        l.add(GenderGroup.FEMALE);
69        l.add(GenderGroup.UNKNOWN);
70        return l.iterator();
71      }
72
73      @Override
74      public boolean isEligible(final SubjectData s) {
75        if (s == null) {
76          throw new IllegalArgumentException("s cannot be null");
77        }
78        return this.gender == GenderGroup.FEMALE && s.isFemale()
79        || this.gender == GenderGroup.MALE && !s.isFemale();
80      }
81
82      @Override
83      public Group newInstance(final Object o) {
84        if (o == null) {
85          throw new IllegalArgumentException("o cannot be null");
86        }
87        return new GenderGroup((Character) o);
88      }
89  }
```

4.5 How to Add a New Parser

The parsing system is described in Section 3.7. If a TAUPE 's user acquires a new eye-tracking system or if the structure of the generated eye-tracking files changes, a developer must add a new parser.

When a new parser is added, it is represented by a new sub-class of the abstract class named `AbstractParser` and this new class must be in the package `laigle.taupe.viewer.parsers.types`. There are three abstract methods to implement:

[**String getName()**] The parser's name is shown to the user when she must choose a parser.

[**String getDescription()**] This methods gives the description of the typical input files required by this new parser. The description is displayed on the panel which allows TAUPE users to choose the parser.

[[**Experiment parse(String eyeTrackerPath, String questionPath, int minDuration)**]] This method performs the actual parsing. The methods `notifyProgressState` and `notifyState` must be used to notify the software about the parsing's progression.

Advices

- The abstract class `AbstractParser` defines a lot of usefull methods to help the parsing of files.
- It is easier to implement the `parse` method following those steps:
  1. Get a `Experiment`'s singleton and set its `minDuration`'s value
  2. If needed, create an instance of `RelevantSlideParser`.
  3. Use the *protected* method `parseQuestionPath` implemented in `AbstractParser` to get the questions from the directory related to the `questionPath`'s value. Then, add those questions to the experiment.
  4. Use the *protected* method `getFiles` implemented in `AbstractParser` to get the files generated by the eye-tracking device (in the directory related to the value of the parameter named `eyeTrackerPath`).
  5. For each of these files, parse its content and add the resulting instances of `SubjectAnswer` to the experiment.

4.6 How to Add a Printer

Printers are described in Section 3.6 and a complete example is given in Section 4.4. A developer who wants create a new type of output for a set of results must implement the interface `IPrinter`. There are as methods to implement as there are children of `AbstractResult`. These `visit` methods typically create per file by result and `ContainerResult.visit` creates a directory for its children. The developer must put the new printer into the package `laigle.taupe.viewer.computation.printer`.

4.7 How to Add an Element to the Preferences

It is easy to add a new element in the preference (described in Section 3.4). A developer must add a new `Argument`'s instance into the attribute named `parameters` of `ConfigurationModel`. Usually, these arguments are defined in the constructor of `ConfigurationModel`. Then, the GUI will automatically take into account of the new argument.

4.8 How to Add an Option for the *Graphical Visualisation*

The *graphical visualisation* system is already described in Section 3.8. If a developer wants to add a new *drawer*, she must create a new class which extends the abstract class `AbstractDrawer` and put it in the package `laigle.taupe.viewer.graphics.options`. The GUI will automatically take into account of the new drawer. There are four methods to implement:

[**String getName()**] This method gives the name that is displayed as the label of its related checkbox in the GUI.

[**String getDescription()**] This method gives the checkbox's *tooltiptext* for this drawer.

[**int getPriority()**] If this drawer's priority equals zero, then the drawer will draw before all the drawers that have a higher priority. The developer must take into account of the priority of each *drawer* to avoid a problem of juxtaposition.

[**void draw(g : Graphics)**] This method obtain information about the answer to draw using the *protected* attribute `AbstractDrawer.answer`. The *protected* attribute `color` can be used in this method.

Let us assume that a developer wants a *drawer* that is able to write the number of fixations of an answer on the left top corner of the panel. She will write the following class named `NumberOfFixationsDrawer`:

```
1  package laigle.taupe.viewer.graphics.options;
2
3  public class NumberOfFixationsDrawer extends AbstractDrawer {
4
5    @Override
6    public String getName() {
7      return "Number␣of␣fixations";
8    }
9
10   @Override
11   public String getDescription() {
12     return "It␣writes␣the␣number␣of␣fixations␣on␣the␣left␣top␣
         corner␣of␣the␣image.";
13   }
14
15   @Override
16   public int getPriority() {
17     return 2;
18   }
19
20   @Override
21   public void draw(Graphics g) {
22     if (g == null) {
23       throw new IllegalArgumentException("g␣cannot␣be␣null");
24     }
25
26     g.setColor(this.color);
```

```
27        g.drawString(answer.getNumberOfFixations()+"␣fixations",
              0, 0);
28      }
29  }
```

5 Compiling

To compile the sources and to generate the `.jar` file, a developer must use the *Apache Ant*[6] tool:

```
1 ant
```

or

```
1 ant run
```

If the developer wants to modify the compilation's preferences, she must modify the file named `build.xml` that describes how the `ant` command is configured. For example, if the sofwate needs a new *library*, the `classpath` attribute must be updated and the `jar` task must contains a new `zipfileset` element.

[6] http://ant.apache.org

6 Test Cases

The test cases related to TAUPE are written with *JUnit*[7] and are stored in the package `test`.
The directory `rsc/tests` contains the files that are parsed and used by the tests. The class named `Initialisation` defines the method `setUp()` that parses the input files. To make the development of tests easier, the test classes can extend this class.
According to the 4^{th} version of *JUnit*, it is recommended to add the annotation `org.junit.Test` to each test method. It is now useless to extend the class `junit.framework.TestCase`.

For example, if a developer wants to write a test that checks whether the value of `Experiment.minDuration` is `0`, she should write the following class:

```
1  package test;
2
3  public class TestExperiment extends Initialisation {
4
5    @Test
6    public void testMinDuration() {
7      Assert.assertEquals(0, this.e.getMinDuration());
8    }
9
10 }
```

[7] http://www.junit.org

7 Licence

7.1 GNU GPL

TAUPE follows the terms of the GNU General Public Licence as published by the Free Software Foundation. A copy of this licence is given in `./COPYING_TAUPE.txt` or can be found at http://www.gnu.org/licenses/gpl.txt.
The following text must be added at the top of each source file in the software.

```
/*
 * This file is part of TAUPE (Thoroughly Analysing the
 * Understanding Programs through Eyesight).
 *
 * TAUPE is free software: you can redistribute it and/or
    modify
 * it under the terms of the GNU General Public License as
    published by
 * the Free Software Foundation, either version 3 of the
    License, or
 * (at your option) any later version.
 *
 * TAUPE is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty
     of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
    the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
    License
 * along with TAUPE.  If not, see <http://www.gnu.org/licenses
    />.
 */
```

7.2 Icons

Here follow the licences related to each icon used in TAUPE .

Directory	File	Licence
icons/	about.png	GNU GPL
icons/	about_small.png	GNU GPL
icons/	add.png	GNU GPL
icons/	clear.png	GNU GPL
icons/	directory.png	GNU GPL
icons/	draw.png	GNU GPL
icons/	exit.png	GNU GPL
icons/	eye.png	Creative Commons[8]
icons/	file.png	GNU GPL
icons/	load.png	GNU GPL
icons/	load_small.png	GNU GPL
icons/	notification0.png	GNU GPL
icons/	notification1.png	GNU GPL
icons/	notification2.png	GNU GPL
icons/	ok.png	GNU GPL
icons/	preferences.png	GNU GPL
icons/	preferences_small.png	GNU GPL
icons/	process.png	GNU GPL
icons/	question.png	GNU GPL
icons/	remove.png	GNU GPL
icons/	save.png	GNU GPL
icons/	ShowGraphicalExperiment_small.png	GNU GPL
icons/	subject.png	GNU GPL
icons/	unknown.png	GNU GPL
icons/command/	AOIMaker.png	GNU GPL
icons/command/	GetCacheSummary.png	GNU GPL
icons/command/	ComputeResults.png	GNU GPL
icons/command/	ShowGraphicalExperiment.png	GNU GPL

[8] Creative Commons(Attribution-NonCommercial 3.0 Unported (CC BY-NC 3.0))

8 Contacts

Team

Alphabetically:

- **De Smet Benoît** Main developer (FUNDP)
  ben.desmet@gmail.com
- **Guéhéneuc Yann-Gaël** Team Leader (Ptidej)
  yann-gael.gueheneuc@polymtl.ca `at` .umontreal.ca
- **Lempereur Lorent** Main developer (FUNDP)
  lorent.lempereur@gmail.com

Links

- **École Polytechnique De Montreal** Software Engineering Department
  http://www.polymtl.ca/gigl/
- **Facultés Notre-Dame-De-La-Paix (FUNDP Namur)** Faculty of Computer Sciences
  http://www.info.fundp.ac.be
- **Ptidej**
  http://www.ptidej.net

Index

References

[1] Y.-G. Guéhéneuc, "Taupe: Towards understanding program comprehension," in *Proceedings of The Conference of the Center for Advanced Studies on Collaborative Research (CASCON'06)*, October 2006, pp. 1 – 13.

[2] R. J. Erich Gamma, Richard Helm and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub Co, 1995.

[3] J. D. from Sun, "Programming With Assertions," http://download.oracle.com/javase/1.4.2/docs/guide/lang/assert.html#usage-conditions, accessed November 28 in 2010.

[4] J. Deacon, "Model-View-Controller (MVC) Architecture," 2009.

Annexe D

Taupe - Guide de l'utilisateur

Facultés Universitaires Notre Dame De la Paix (FUNDP Namur)
Faculté d'Informatique

École Polytechnique de Montréal
Département de Génie Logiciel

Team Leader: Yann-Gaël Guéhéneuc

TAUPE v2.0 - User's Guide

Benoît De Smet
Lorent Lempereur

May 10, 2011

Contents

List of Figures

1 Introduction

TAUPE[1] (Thoroughly Analyzing the Understanding of Programs through Eyesight [1]) is a software designed by the *Ptidej team*[2] to import data from eye-trackers (as described in Section 6) and to allow the execution of various algorithms (descibed in Section 3) on the collected data.

TAUPE was originally designed to compare the way of looking, analysing, and reading of subjects of UML diagrams. A subject was asked different questions about her understanding of the diagrams and had to complete a set of maintenance tasks on it. Her eye's movements were recorded using an eye tracking device. These eye mouvements must be analysed by researchers to assess the comprehension process of the subject. Some softwares in the field of data analysis already existed but none was open source so it was really difficult to run customized algorithms on the collected data.

This document contains the user's documentation and all information needed to use TAUPE. All the information concerning the development and the implementation of extensions for the software can be found in the *Developer's Guide*[3].

This guide concerns the version `2.0` of TAUPE, last version currently available (May 10, 2011).

A complete description of the software including this guide and the developer's guide can be found on the ptidej website[4].

[1] "Taupe" means "mole" in French and is pronounced 'tOp.
[2] `http://www.ptidej.net/team`
[3] Available via `http://www.ptidej.net/research/taupe/`
[4] `http://www.ptidej.net/research/taupe/`

2 Vocabulary

The following are terms used regulary all trough this guide with specific meanings:

Gaze: The gaze is typicaly "where we are looking" this is the data recorded by the eye-tracker.

Fixation: A fixation is a position of the eye during a gaze.

Saccade: A saccade is a movement of the eye between two fixations.

Experiment: An experiment represents the whole system (It is a set of questions, their answers, the fixations, the saccades, the area of interest ...) that we want to analyse with TAUPE. When the user is using the software, this one handles up to only one experiment at a time.

Question: A question is related to one image (usually an UML class diagram) associated with a task that the subject has to do on the diagram.

Subject: A subject is a person who answered a set of questions during the duration of an experiment. It is defined by a set of characteristics like its name or its level of study. Each subject is linked to a file that is generally generated by an eye-tracking device and that contains the whole set of data about the subject's answers.

Group: A group is a set of subjects. A group is defined by its type and characteristics of this one. For example groups that classify the subjects based on their UML knowladge, gender, desgin patterns knowladge...

Area of Interest: An area of interest is an area on a question's image. This area can be relevant or not to a task (relevance is specified in the corresponding `.aoi`). It also can be "ignorable", it means that the system will not take account of this area. Areas are defined in a text file (`AOI files`) for each question. For example, if a question is related to an image called `foo.png`, then the file that defines the areas is named `foo.aoi`. The file structure is defined in Section 5.

Answer: An answer is what a specified subject has answered to a specified question. This answer can be correct or not (determined by the researchers) and is related to a set of fixations and a set of saccades. All the answers for a subject are in the same `.subject` file (see Section 5).

Q**:** The set of questions.

A**:** A set of answers.

F**:** A set of fixations.

S**:** A set of saccades.

AOI**:** A set of area of interest.

F_i**:** Set of fixations in the answer i ($i \in A$) related to the subject's answer i without considering the fixations that are in a "ignorable" AOI.

S_i**:** Set of saccades in the answer i ($i \in A$).

A_i**:** The set of answers related to the question i ($i \in Q$).

$AORI_i$**:** Set of areas of relevant interest related to the question i ($i \in Q$).

$AOII_i$**:** Set of areas of irrelevant interest related to the question i ($i \in Q$).

AOI_i**:** Set of areas of interest related to the question i ($i \in Q$). This set can be calculated with

$$AOI_i = AORI_i \cup AOII_i$$

3 Algorithms

With the current version of TAUPE (2.0), different algorithms are available. The results generated by those algorihms are often sorted by group (ex: Beginners, experts, male, female...) of subjects and some statistics (average, standard deviation...) are sometimes realised.

3.1 Fixations Statistics

As explained in [2], "A fixation algorithm must produce fixations that meet certain minimum characteristics. The center of a typical fixation is within 2-3^o from the observed target object [3] and the minimum processing duration during a fixation is 100-150 ms [4]."
This algorithm provides a set of algorithms related to the fixations:

$t : F \rightarrow \mathbb{N}$**:** The specified fixation's duration (in milliseconds).

$fix : AOI \times A \rightarrow F^n$**:** The fixations in a specified area of interest for a specified answer.

$surface : AOI \rightarrow \mathbb{N}$**:** The specified AOI's surface (in pixels^2)

$FAORI_i$**:** Set of fixations in a Relevant AOI for the answer i ($i \in A_j$) as:

$$FAORI_i = \bigcup_{k \in AORI_j} fix(k, i)$$

$FAOII_i$**:** Set of fixations in an Irrelevant AOI for the answer i ($i \in A$) as:

$$FAOII_i = \bigcup_{k \in AOII_j} fix(k, i)$$

$TAOI_i$**:** Total duration (in milliseconds) of the fixations in a AOI for the answer i ($i \in A_j$).

$$TAORI_i = TAORI + TAOII$$

$TAORI_i$**:** Total duration (in milliseconds) of the fixations in a Relevant AOI for the answer i ($i \in A_j$).

$$TAORI_i = \sum_{f \in FAORI_i} t(f)$$

$TAOII_i$**:** Total duration (in milliseconds) of the fixations in an Irrelevant AOI for the answer i ($i \in A_j$).

$$TAOII_i = \sum_{f \in FAOII_i} t(f)$$

3.1.1 Total Number of Fixations

As mentionned in [2], "The number of fixations overall is thought to be negatively correlated with search efficiency". The number of fixations for a subject's answer i ($i \in A$) is here $\#F_i$.

3.1.2 Number of Fixations per Type of Areas of Interest

There exists currently three kinds of areas of interest: *Relevant*, *Irrelevant*, and *Ignorable*. This latter will not be considered, all the fixations in a "ignorable" aoi will be ignored in all algoritms concerning the aoi. "More fixations on a particular area indicate that it is more noticeable, or more important, to the viewer than other areas." [5]
Therefore, TAUPE makes the distinction between the relevant areas ($\#FAORI_i$) (according to the specified question) and the areas that the subject should not look (irrelevant) ($\#FAOII_i$).

3.1.3 Fixation Duration

According to [6], the duration of the fixations about a specified area can have two different meanings:

1. A longer fixation duration means that the subject has some difficulty to extract information [7].
2. A longer fixation duration means that the object is "more engaging in some way"[8].

3.1.4 Normalized Fixations per Area of Interest

This metric $NORM\_RATE$ represents the ratio between the normalized number of fixations in an AORI and the normalized number of fixations in an AOII[9]. It is used to assess the subject's effort. The $NORM\_RATE_i$ for a subject's answer i ($i \in A_j$) is described as:

$$NORM\_RATE_i = \frac{\frac{\#FAORI_i}{\#AORI_j}}{\frac{\#FAOII_i}{\#AOII_j}}$$

3.1.5 Overall Fixation Rate overall

Not implemented yet. "This metric is closely related to fixation duration. Since the time between fixations (typically short duration saccadic eye movements) is relatively small compared with the time spent fixating, fixation rate should be approximately the inverse of fixation duration." [10]

3.1.6 IN AORI / IN AOII

Not implemented yet. This is the ratio between the number of fixations in the areas of relevant interest and in the areas of irrelevant interest. This metric excludes the fixations that are out of an area of interest. It is used to assess the effort's relevance. This ratio IN_OUT_i for a subject's answer i is defined as:

$$IN\_OUT_i = \frac{\#FAORI_i}{\#FAOII_i}$$

3.1.7 On-target/All-target

Not implemented yet. This metric "can be defined by counting the number of fixations falling within a designated area of interest, then dividing by the all fixations. This is a content-dependent efficiency measure of search, with smaller ratio indicating lower efficiency"[2]. This

metric ON_ALL_i for an area of interest i about a subject's answer j is calculable as:

$$ON\_ALL_i = \frac{\#fix(i,j)}{\#F_j}$$

3.1.8 Post-target Fixations

Not implemented yet. "The number of *post-target fixations*, or fixations on other areas, following target capture, can indicate the target's meaningfulness to a user. High values of non-target checking, following initial target capture indicate target representations with poor meaningfulness or visibility"[2]. TAUPE computes this metric $POST\_TARGET_{ij}$ for an area i about a subject's answer j with the following formula:

$$POST\_TARGET_{ij} = \#FAORI_j + \#FAOII_j - \#fix(i,j)$$

3.1.9 Spatial Density

"[c]overage of an interface due to search and processing may be captured by the spatial distribution of [fixations]. [...] The [image] can be divided into grid areas either representing specific objects of physical screen area. [...] The *spatial density* index [is] equal to the number of cells containing at least one [fixation], divided by the toal number of grid cells. [...] A smaller spatial density indicated more directed search, regardless of the temporal [fixation] sampling order."[2] The cell's size used in TAUPE is 64x64px (this value can be changed see the Developper's guide).

3.2 Saccades Statistics

This set of metrics can be used to compute statistics about saccades.

3.2.1 Transition Density

The transition density is based on the notion of *transition matrix* and *link analysis*. "[F]requent transitions from one region of a display to another indicates inefficient scanning with extensive search. [...] The transition matrix is a tabular representation of the number of transitions to and from each defined area."[2]. A cell (part of the grid areas either representing specific objects of physical screen area) is filled if a saccade starts of ends in its area. With the *transition matrix*, it is possible to compute its density.

$$TRANSITION\_DENSITY_i = \frac{\sum_{x \in C} isFilled(x)}{\#C}$$

where:

C_i is the set of cells in the transition matrix of the subject's answer i ($i \in Q$).

$isFilled : C \to \{0,1\}$**:** is a function returning 1 if the specified cell is filled and 0 otherwise.

3.3 Convex Hull Computing

This algorithm can be used to compute the convex hull[5] of all the fixations for each group defined in TAUPE. FIGURE 1[6] displays the convex hull as a imagining band stretched open to encompass whole set of fixations. TAUPE uses an algoritm described in [11] to compute the convex hull of the whole set of fixations.

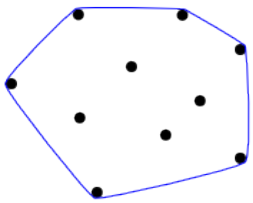


Figure 1: Convex Hull (in blue)

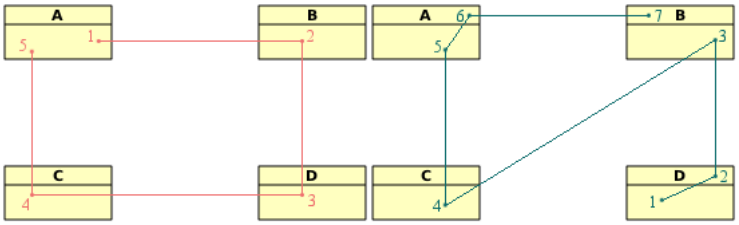
This algorithm generates the convex hull for each subject's answer. Moreover, the software builds the convex hulls for a series of percentages. These percentages corresponds to the percentage of keeped fixations (eg: 50% means that only half of the fixations are used, the longest ones). Currently, this series is defined on the range [5%...100%] with an interval of 5%.

3.4 Visual Path Computing

This algorithm can be used to compute the differences between the *visual path* of two subjects' answers.
A *visual path* is a serie of visited areas of interest sorted chronologically. An area is visited if there is a fixation in it.
For example, if a question contains four areas of interest `A`, `B`, `C` and `D`, then two possible *visual paths* would be `ADCABC` or `DABC`. In the case of FIGURE 2, each area is related to a class in a simple UML class diagram), then a possible *visual path* would be `ABDC` (FIGURE 2a) or `DBCAB` (FIGURE 2b). However, a same area of interest that is visited twice consecutively (two consecutives fixations in the same area of interets) is considered only once. So, while the FIGURE 2b's visual path could be `DDBCAAB`, it is actually `DBCAB`.

The difference between two visual paths is computed using an *edit distance* algorithm: the *Levenshtein algorithm*. The Levenshtein algorithm compares two strings. TAUPE uses its own adaption of the algorithms to handle a list of area of interest.
TAUPE compares two subjects' visual paths according to the percentage of fixations taken [5%...100%] with a granularity of 5%. For example, if the percentage of fixations considered is 75%, then the choice of fixations will be based on the fixations' duration (the 75% longest fixations will be taken).

[5]The convex hull or convex envelope for a set of points X in a real vector space V is the minimal convex set containing X.
[6]Source: http://commons.wikimedia.org/wiki/File:ConvexHull.png



(a) ABDC (b) DBCAB

Figure 2: The visual path

3.5 Correctness Computing

This computation method can be used to generate a summary (for each group) about the correctness of the answers provided by the subjects via the `.subject` files (5.1).

3.6 Time Computing

This algorithm can be used to compute statistics about the time spent on each question for each group. Thus, this module does not compute statistics about the fixations' duration, the time spent in a specified area of interest...

3.7 Discussion

- Two scan paths can have the same *convex hull* (section 3.3) and the same *spatial density* (section 3.1.9) but they can have a different *transition density*. A better scan path would have a lesser dense *transition matrix*.
- Typically, longer is an answer, lesser the subject's efficiency. However, [12] thinks that in the case of the reading, this metric is not exact because of the inconstant speed reading of the subjects.

4 How Tos

4.1 How to Launch Taupe

TAUPE is provided in a `.jar` file. To run TAUPE, just open a command line interface and type:

```
1 java -jar TAUPE.jar
```

or launch the JAR file from your file browser if your system allows it. It is possible that you need to increase the amount of memory allowed to Java if you have to load a lot of data. You do this by adding the `-Xms`A`m -Xmx`B`m` parameter, where A is the initial amount of memory (in MB) and B the maximal amount. For example:

```
1 java -Xms32m -Xmx1024m -jar TAUPE.jar
```

Please note that de `.jar` file should not be renamed, the file name must be *TAUPE.jar*.

4.2 How to Change the Software's Preferences

To set up the preferences, simply click on `File` → `Preferences` in the menu or press `Ctrl + P` . The configuration dialog box (FIGURE 3) will pop up.

Default directory for the AOI output generation: This directory is used by the AOI maker (FIGURE 7 and Section 4.5) to store the generated `.aoi` files (see 5.2).

Result directory: This directory is used to store the files generated by TAUPE which contains all the results. This is the output directory.

Data directory for experiments: This directory is used to load the files from the eye-tracker. The directory must contain the data from the eye-tracker (ex: files `.out` for *GazeTracker*) **AND** their corresponding `.subject` files as described in the section 5.1. The files must have the same names; for example: `Subject01.out` and `Subject01.subject`, `Subject02.out` and `Subject02.subject`, and so on. This is a input directory.

Data directory for diagrams: This is the directory that contains the images to load (ex: the diagrams images). Each image listed in the files from the eye-tracker must be in this directory. Also each image file (`.jpg` or `.png` file) must be accompagned by a `.aoi` file as described in Section 5.2. The image file and the AOI file must have the same name; for example: `JFreeChart_MD_Q1.png` and `JFreeChart_MD_Q1.aoi`.

4.3 How to Load Data

Before loading data into TAUPE the preferences must be set up as described in section 4.2.

1. To load data from an eye-tracker, in the main window, shown in FIGURE 5, click on `File` → `Load` in the menu (or press `Ctrl + L`). Select the parser according to the type of data you want to load in the dialog box FIGURE 4.

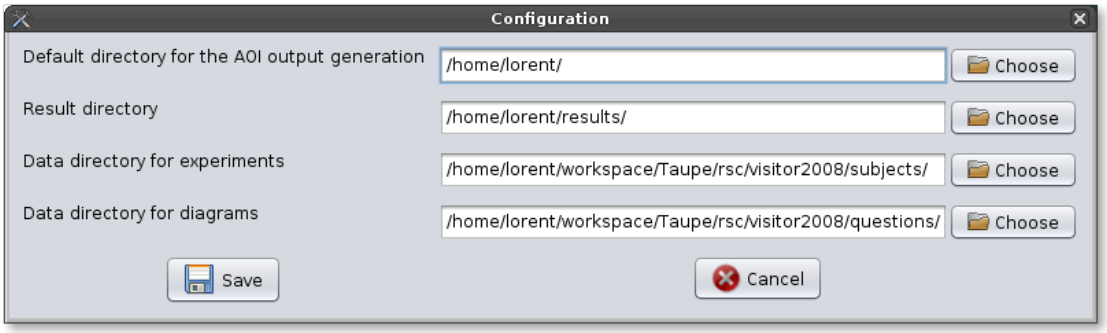


Figure 3: The preferences window

2. Choose the directory containing the data from the eye-tracker as described in Section 4.2 (*Data directory for experiments*). For the *GazeTracker* parser a file named `relevant.slides` is also needed in the directory that lists the images that should be loaded in TAUPE. All images not listed in this file will be ignored and will not be loaded. `relevant.slides` can be used if you need to ignore some data in some slides (e.g., if there is a white screen between each diagram). The structure of the `relevant.slides` file is explained in Section 5.3.

3. The *duration* (in milliseconds) parameter is used to specify the minimal amount of time that a fixation should have to be considered as such.

4. Press the `OK` button, the parsing starts. The loading could take some times depending on the amount of data contained in the eye-tracker's files and the number of subjects.

Figure 4: The parser chooser dialog box

4.4 How to Display Graphical Data

To display some data (saccades, convex hull, fixations...) on the panel, must load data as described in Section 4.3. Once the data is loaded in TAUPE, it is possible to view the loaded data.

1. Simply launch the visualisation tool shown on FIGURE 6 by clicking on the `Visualisation Tool` button an then on the `Execute` button on the right panel. The experimentaiton GUI is divided in two main parts. The panel on the left is used to display the image and data and the right panel is used to choose which information to be shown.

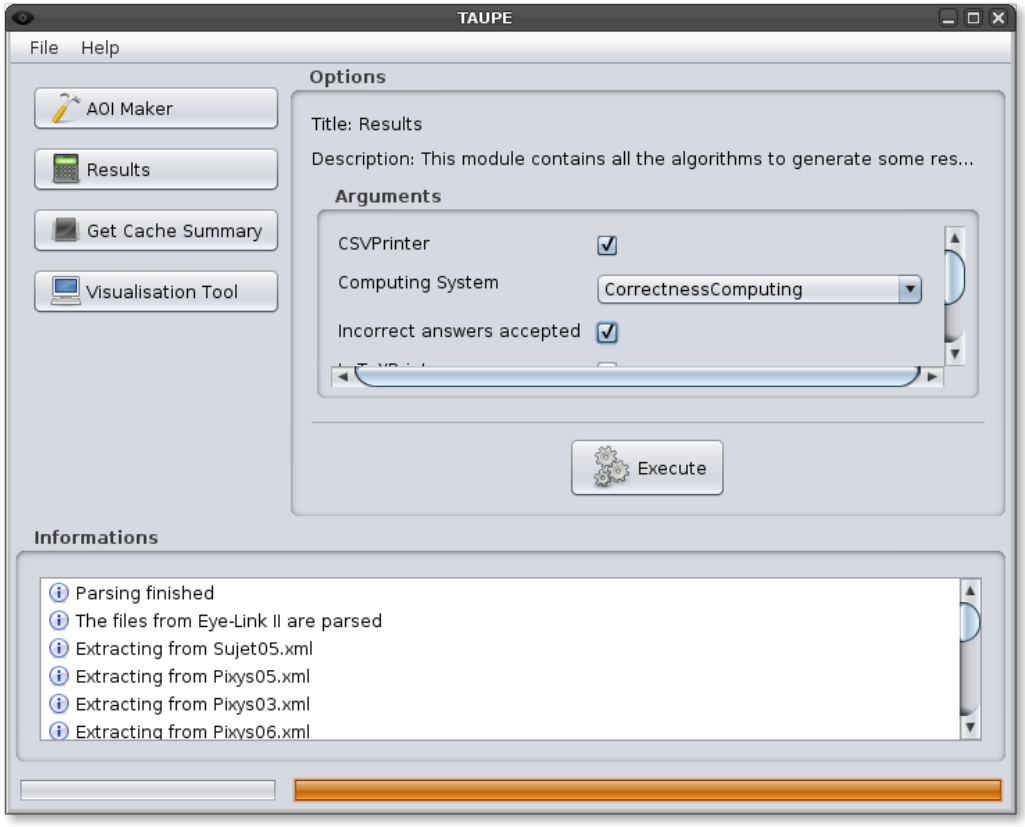


Figure 5: The TAUPE's main window

2. To display data simply tick the corresponding checkbox and choose its color. By pressing the `Draw` button the selected information will appear on the left panel. The image and the corresponding subject can be selected via the drop boxes in the top of the right panel.

3. The slider at the bottom of the right panel can be used to set the percentage of displayed fixations, for example to keep the longest fixations.

4. Clicking on a fixation will show its characteristics (duration, coordonates...) in the text area above the `Draw` button. Under this button, a progress bar can be found which show you the progression of the current drawing.

4.5 How to Create a Set of Areas of Interest

An area of interest is a polygon on an image taht TAUPE can use to distinguish fixations. The AOI maker shown on FIGURE 7 can be used to easily create a set of areas of interest on an image. To start the AOI maker, click on the AOI maker button and then on the `Execute` button. **This module saves you the effort to write an AOI file manually.**

1. To create a set of areas of interest it is needed to have an image loaded. To do so, click on `File` → `Load` or press `Ctrl + O`. In the bottom section the image file to be loaded

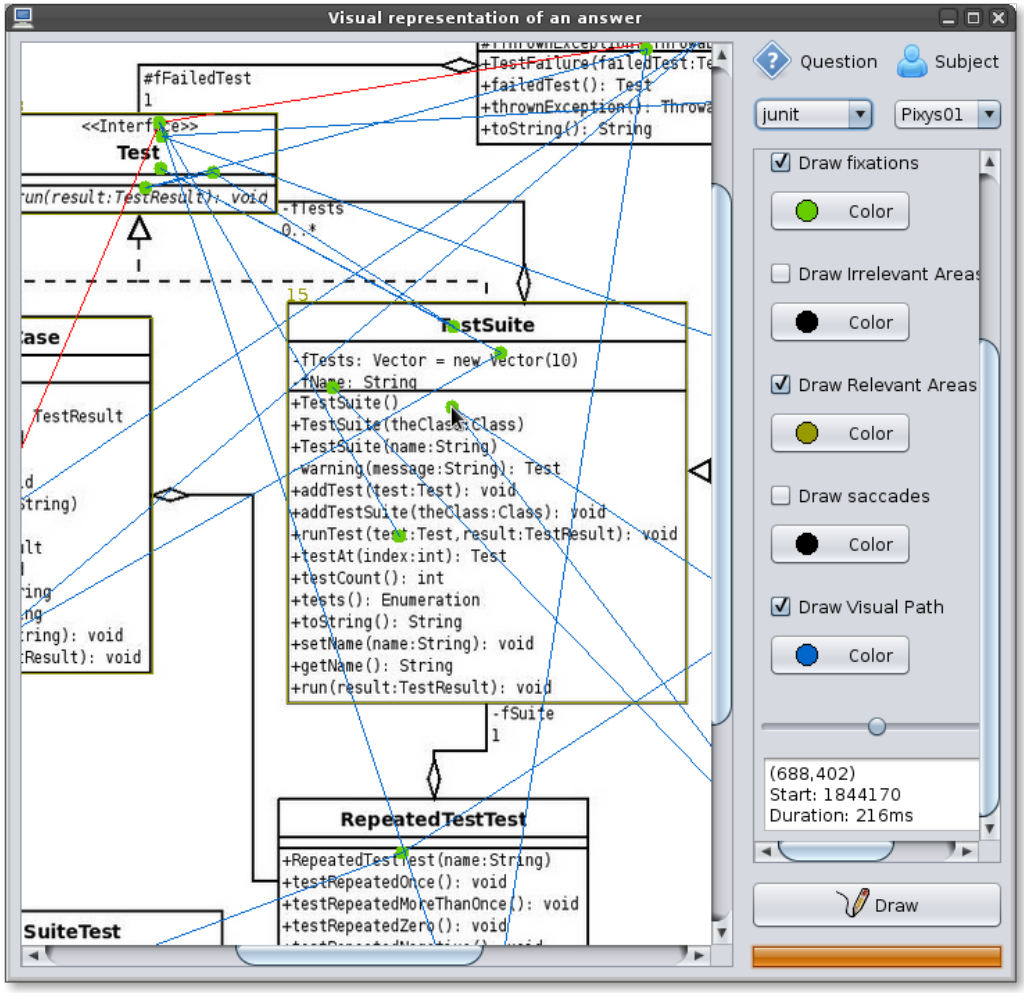


Figure 6: The visualisation tool

can be choosen. You can also load an AOI file if you want to edit it. Once the image is loaded, it is possible to draw a shape directly on the panel by left clicking on different points around the area that is of interest.

2. Once the area of interest is drawn, it is possible to add it to the set of AOI by clicking on `Area Of Interest` → `Add` or by pressing `Ctrl + A`. Some information can be entered about the area of interest such as its identifier (a unique number used to identify the AOI on this image), its name, and its relevance. There are currently three available levels of relevance: *Ignorable*, *Relevant* and *Irrelevant*. *Ignorable* means that the fixations in this AOI will not be taken in account for all the algorithms described in Section 3.

3. Press the `Add` button to add the area of interest. It is, of course, possible to add multiple area of interest on a single image. The `Clear` button can be used to remove the points which are not saved yet.

4. To save the AOI file click on `File` → `Save` or press `Ctrl + S`, then select the place and name of the file you want to save and press `Save`. To remove an AOI simply click

on `Area Of Interest` → `Remove` or press `Ctrl + R`, then select the AOI you want to remove and press the `Remove` button.

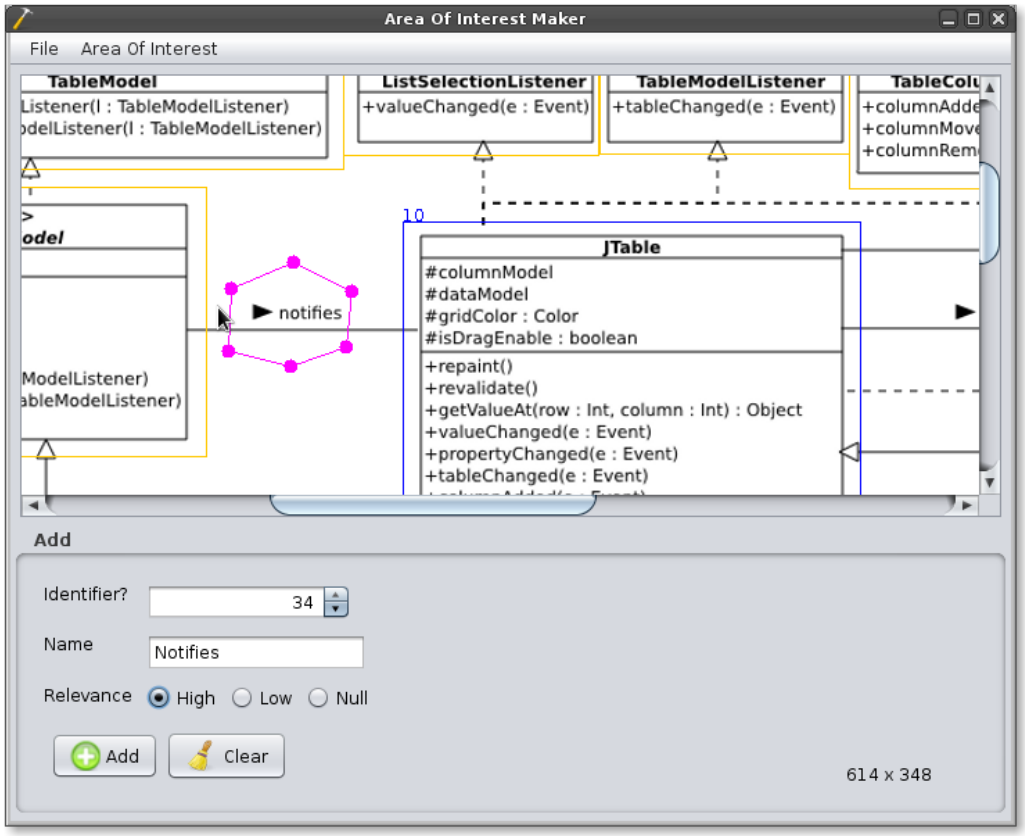


Figure 7: The AOI maker interface

4.6 How to Execute an Algorithm on some Data

To execute an algorithm on the loaded data click on the result button and then select the algorithm to apply on the data and set up its parameters. For each algorithm it is needed to select its printer and its result directory in which the resulting files will be stored. Moreover, it is needed to choose whether the wrong answers have to be taken in account or not (if the fixations corresponding to a task which was not correcly done by the subjects should be used or not).
Once the computation is done the results can be accessed in the directory previously specified.

5 Input Files Structures

5.1 `.subject` files

5.1.1 General description

A `.subject` file must contain the tag `Subject:` followed by the characteristic of the user followed by its value; for example, `Subject:Expert:1` means that the subject associated with this file is considered as an expert. Actually, there are six possible characteristics implemented in TAUPE:

- `UMLKnowledge:` the UML level of the subject (in [0..2]).
- `DPKnowledge:` the Design Pattern knowledge of the subject (in [0..1]).
- `Gender:` the gender of the subject (`M` — `F`) *(Male – Female)*.
- `Morethan20classes:` whether the subject has already worked on a UML diagrams that contained more than 20 classes (`1`) or not (`0`)
- `Expert:` whether the subject is considered as an expert (`1`) or not (`0`).
- `StudyLevel:` the level of study of the subject (`U` – `B` – `M` – `D`) *(Unknown – Bac – Master – PhD)*.

A `.subject` file must contain the tag `Answer:` followed by the question (name of the image file without its extension) followed by `1` if the subject answered correctly to the question, or `0` otherwise.
ex: `Answer:quickuml:1` means that the subject answered correctly the question asked on the diagram `quickuml`.

5.1.2 Example

Here follows an example of a correct `.subject` file:

```
1 Subject:UMLKnowledge:2
2 Subject:DPKnowledge:1
3 Subject:Gender:M
4 Subject:Morethan20classes:1
5 Subject:Expert:1
6 Subject:StudyLevel:D
7 Answer:junit:1
8 Answer:quickuml:1
9 Answer:argouml:1
```

5.1.3 EBNF

```
[<SubjectCaract> | <AnswerCaract>]*
<SubjectCaract>::= "Subject:"<caracteristic>":"<value> <EOL>
<caracteristic>::= string
<value>::= char
<EOL>::= EndOfLine
<AnswerCaract> ::= "Answer:"<question>":"<validResponse>
<question>::= string
<validResponse> ::= "1" | "0"
```

5.2 `.aoi` files

5.2.1 General description

An area of interest is an area on a question's image. This area can be relevant or not to a task (relevance is specified in the corresponding `.aoi`). It also can be "ignorable", it means that the system will not take account of this area. Areas are defined in a text file (`AOI files`) for each question. For example, if a question is related to an image called `foo.png`, then the file that defines the areas is named `foo.aoi`.

5.2.2 Example

```
10 NULL Question (17,19) (250,19) (250,65) (17,65)
1 AOI Event (524,83) (727,83) (727,175) (524,175)
2 AORI EventListener (835,83) (835,167) (1015,167) (1015,83)
3 AOI ActionListener (155,215) (395,215) (395,327) (155,327)
7 AORI ChartChangeListener (395,215) (395,310) (613,310) (613,215)
9 NULL ChartProgressListener (288,80) (330,58) (390,63) (390,116) (348,119) (302,107)
```

5.2.3 EBNF

```
[<id> <type> <name> <coordonate> <coordonate> <coordonate>+ <EOL>]*
<id>::= integer
<type> ::= NULL | AOI | AORI
<coordonate> ::= "("integer "," integer")"
<EOL> ::= EndOfLine
```

5.3 `relevant.slides` file

5.3.1 General description

This file is used to select the relevant images to load on TAUPE. It is composed by a set of image names without the extension (for lisibility reasons). All images not listed in this file will be ignored by the parser.

5.3.2 Example

```
1 Swing_MD_Q1
2 Swing_MD_Q2
3 ...
4 Swing_MVP_Q6
5 JFreeChart_MD_Q1
6 JFreeChart_MD_Q2
7 JFreeChart_MD_Q3
8 ...
9 JFreeChart_MVP_Q6
```

5.3.3 EBNF

```
1 [<slide> <EOL>]*
2 <slide> ::= String
3 <EOL> ::= EndOfLine
```

6 Eye-tracking systems

This section briefly describes the file structure of the output files from the eye-trackers systems. For more information, please refer to the manual of the corresponding eye-tracker. Typically, an output file is a set of fixations and saccades listed in a text file. There are different output formats but the structure is basicly the same: A time (sometimes a timestamp in milliseconds), a type (fixation or saccade), the coordonates on the screen (x and y) and the duration. Some other information such as the pupil size can also be present in the file.

6.1 Eye-link II

Eye-link's software generates `.edf` files. TAUPE cannot read directly this type of files. A tool named `edf2xml` can be used to generate `.xml` files from the `.edf` files. The `.xml` files can be loaded in TAUPE. This eye-tracker provides some information about the saccades' *peak velocity* and *amplitude* but TAUPE do not currently use this data.

6.2 GazeTracker

GazeTracker software can generate `.out` files that can be loaded in TAUPE using the corresponding parser (4.3). A screencast on how to export data from GazeTracker is available at: http://www.ptidej.net/research/taupe/.

7 Shortcuts

This section is a summary of all shortcuts which are available in TAUPE.

7.1 Main Window

`Ctrl + O`: Opens the parser chooser dialog box (see FIGURE 4).

`Ctrl + P`: Opens the preferences window (see FIGURE 3).

`Ctrl + A`: Opens the *About* window.

7.2 AOI Maker

`Ctrl + O`: Allows to load a image file and a AOI file.

`Ctrl + S`: Allows to save the set of AOI created.

`Ctrl + A`: Allows to add the recently drawn AOI to the set of AOI.

`Ctrl + R`: Allows to remove an AOI.

Index

References

[1] Y.-G. Guéhéneuc, "Taupe: Towards understanding program comprehension," in *Proceedings of The Conference of the Center for Advanced Studies on Collaborative Research (CASCON'06)*, October 2006, pp. 1 – 13.

[2] J. H. Goldberg and X. P. Kotval, "Computer interface evaluation using eye movements: methods and constructs," *International Journal of Industrial Ergonomics*, vol. 24, no. 6, pp. 631 – 645, 1999.

[3] G. Robinson, "Dynamics of the eye and head during movement between displays: A qualitative and quantitative guide for designers," *Human Factors*, vol. 21 (3), pp. 343 – 352, June 1979.

[4] P. Viviani, "Eye movements and their role in visual and cognitive processes," *Elsevier Science*, 1990.

[5] A. Poole and L. J. Ball, "In search of salience: A response time and eye movement analysis of bookmark recognition," *People and Computers XVIII-Design for Life: Proceedings of HCI 2004*, 2004.

[6] M. Just and P. A. Carpenter, "Eye fixations and cognitive processes," *Cognitive Psychology*, vol. 8, no. 4, pp. 441 – 480, 1976.

[7] P. M. Fitts, R. E. Jones, and J. L. Milton, "Eye movements of aircraft pilots during instrument-landing approaches," *Aeronautical Engineering Review*, vol. 9, no. 2, pp. 24–29, 1950.

[8] A. Poole and L. J. Ball, "Eye tracking in human-computer interaction and usability research: Current status and future prospects," *Ghaoui, Claude (Ed.). Encyclopedia of Human Computer Interaction*, 2006.

[9] S. Jeanmart, "Evaluation de l'impact d'un patron de conception sur la compréhension et la maintenance de programmes - une expérimentation par un système d'eye-tracking," Master's thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 2008.

[10] R. J. K. Jacob and K. S. Karn, "Commentary on section 4: Eye tracking in human-computer interaction and usability research: Ready to deliver the promises," *Work*, no. 1905, 1958.

[11] G. T. Heineman, G. Pollice, and S. Selkow, *Algorithms in a Nutshell (In a Nutshell (O'Reilly))*. O'Reilly Media, 2008.

[12] S. Yusuf, H. Kagdi, and J. I. Maletic, "Assessing the comprehension of uml class diagrams via eye tracking," in *Proceedings of the 15th IEEE International Conference on Program Comprehension*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 113–122.

ANNEXE E

Diagrammes de l'expérience

E.1 JFreeChart / *Model-Delegate*

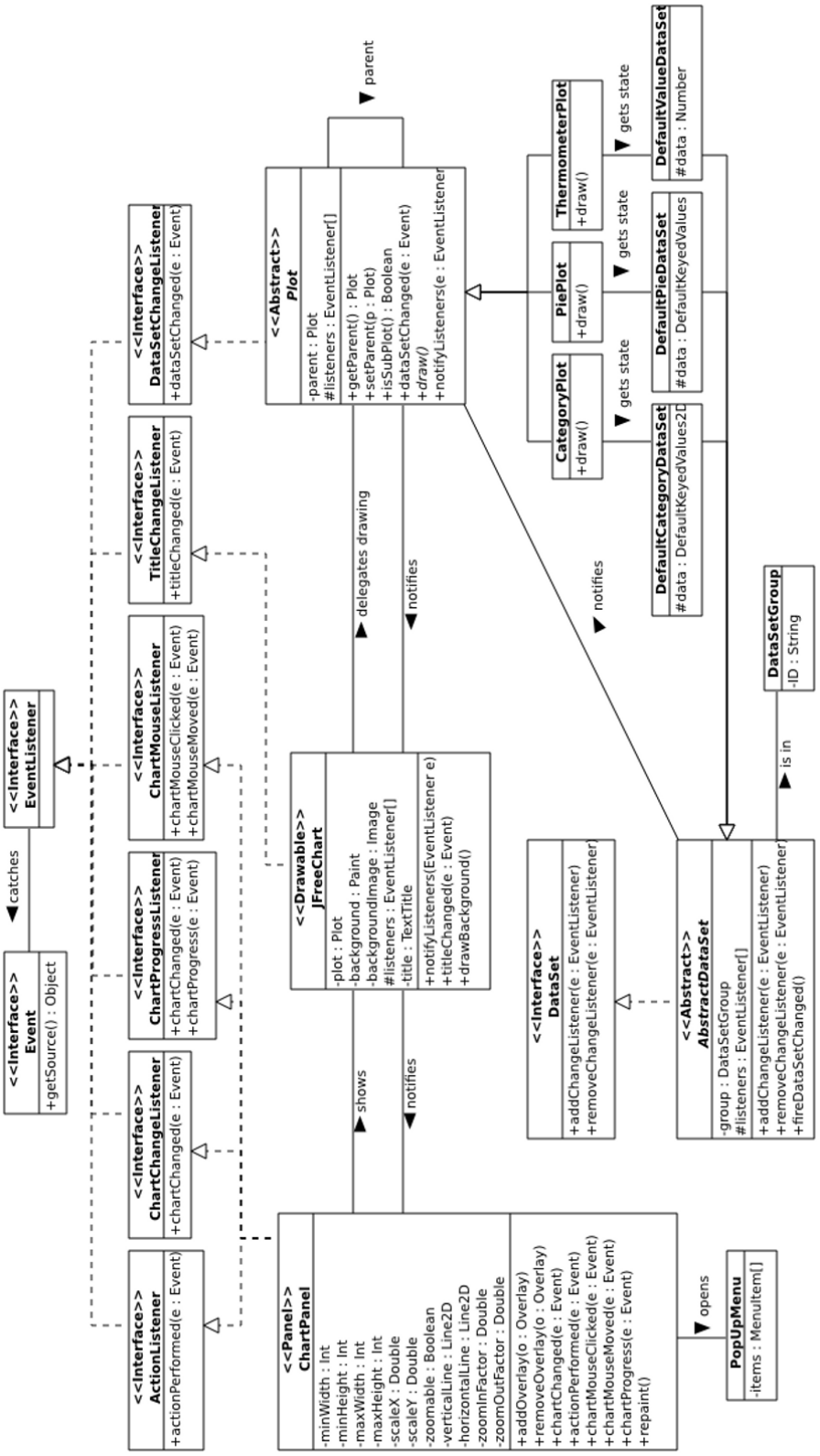


FIGURE E.1 – JFreeChart / *Model-Delegate*

E.2 JFreeChart / Modèle-Vue-Contrôleur

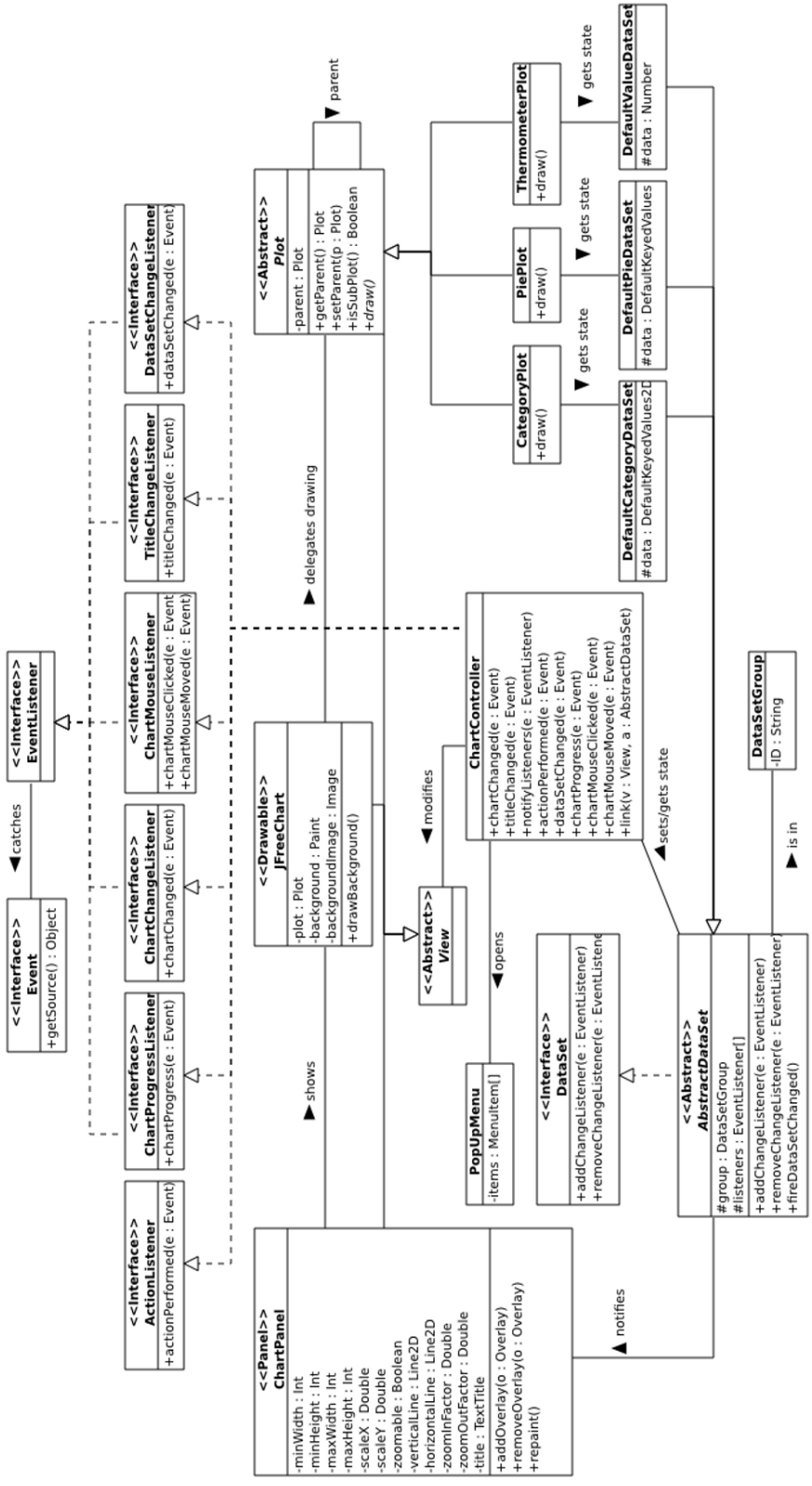


FIGURE E.2 – JFreeChart / Modèle-Vue-Contrôleur

E.3 JFreeChart / Modèle-Vue-Présentateur

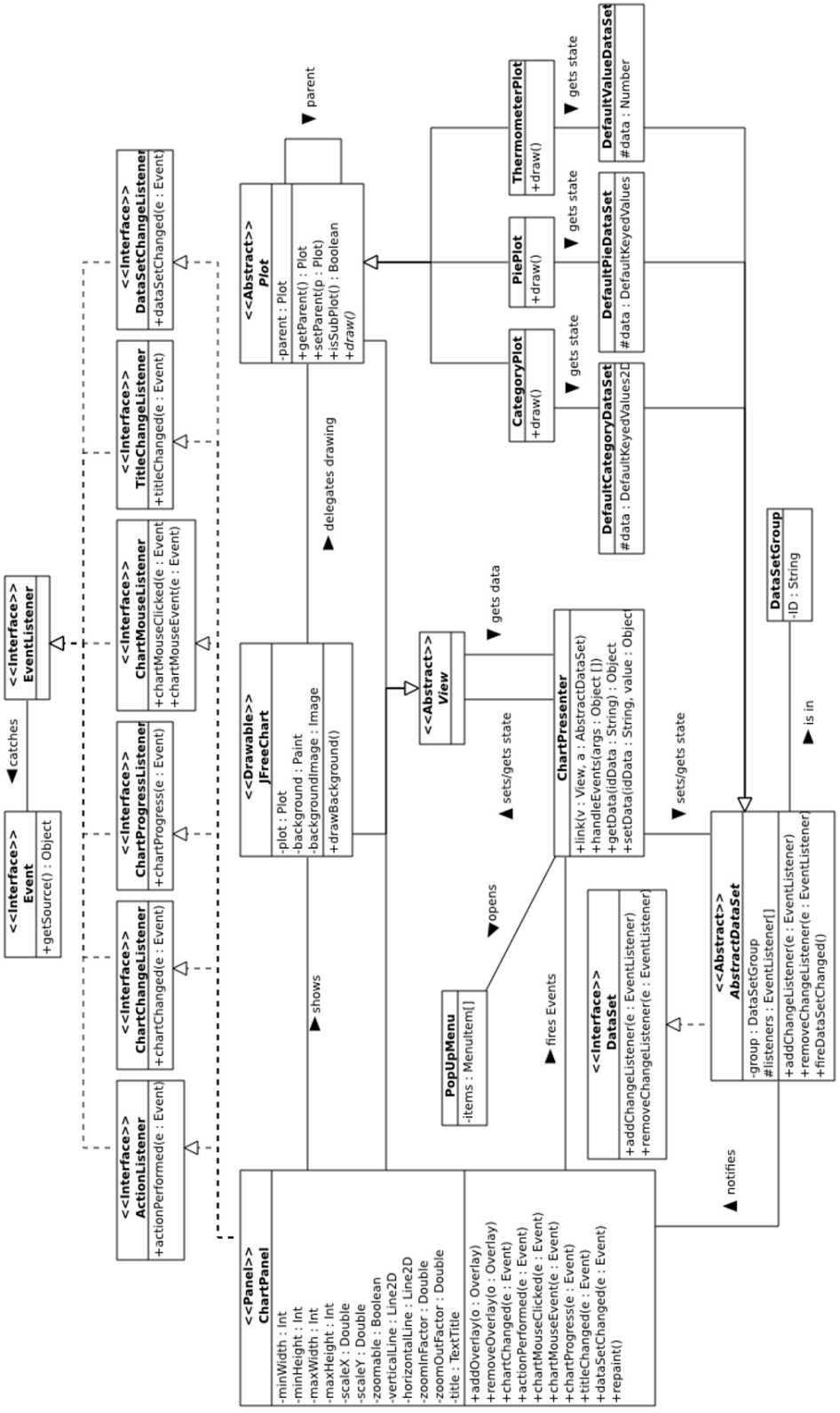


FIGURE E.3 – JFreeChart / Modèle-Vue-Présentateur

E.4 Swing / *Model-Delegate*

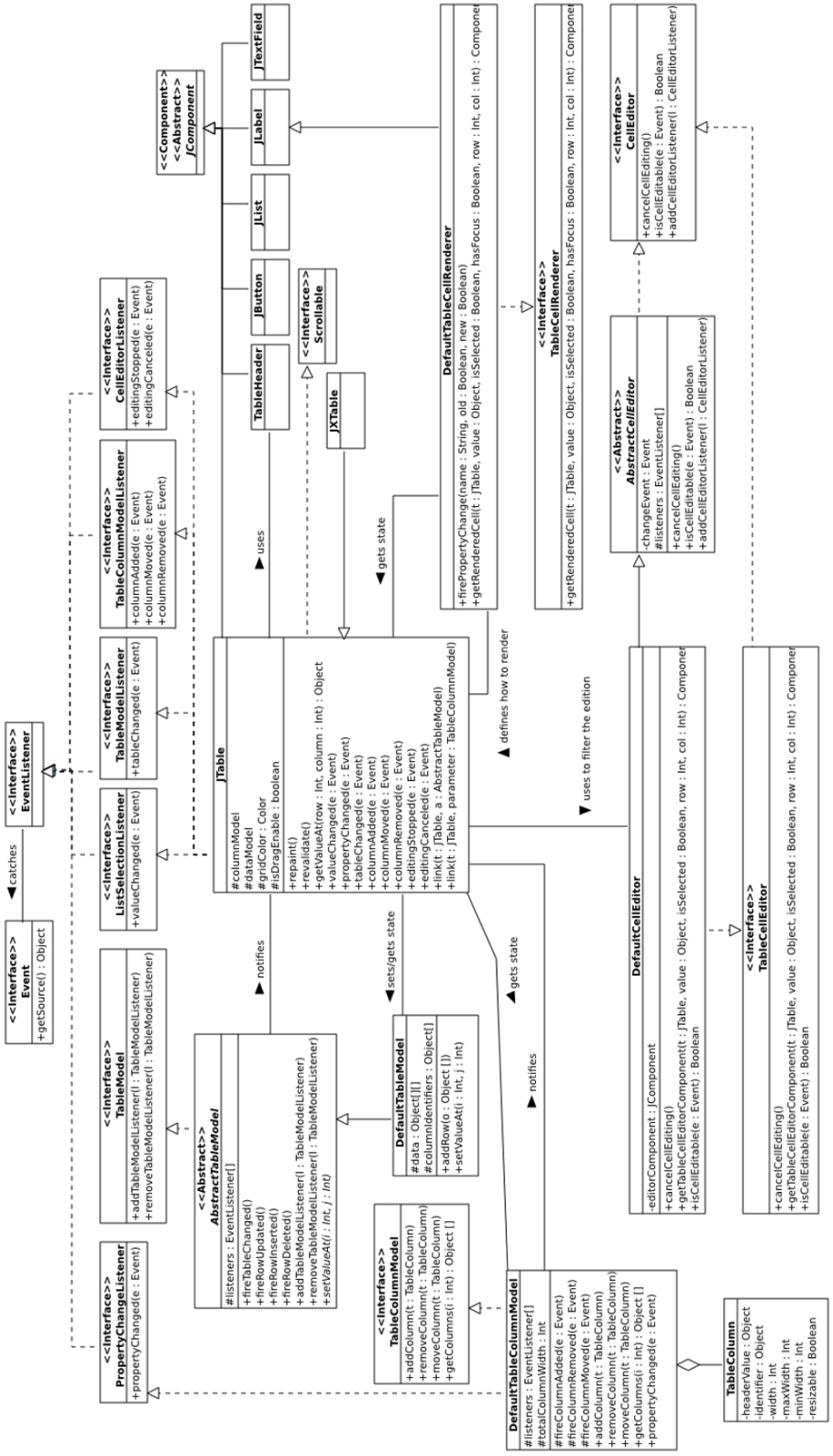


FIGURE E.4 – Swing / Model-Delegate

E.5 Swing / Modèle-Vue-Contrôleur

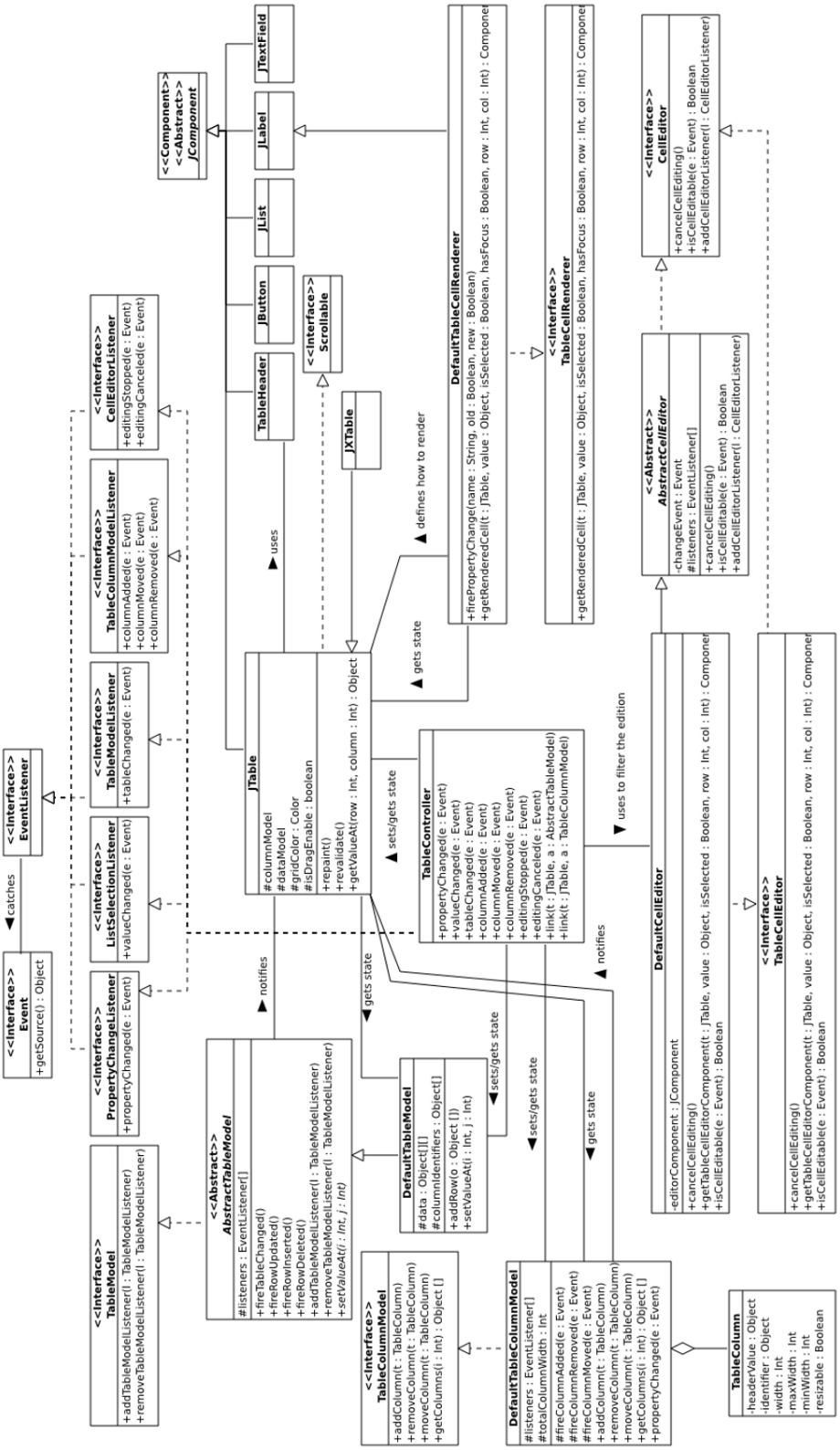


FIGURE E.5 – Swing / Modèle-Vue-Contrôleur

E.6 Swing / Modèle-Vue-Présentateur

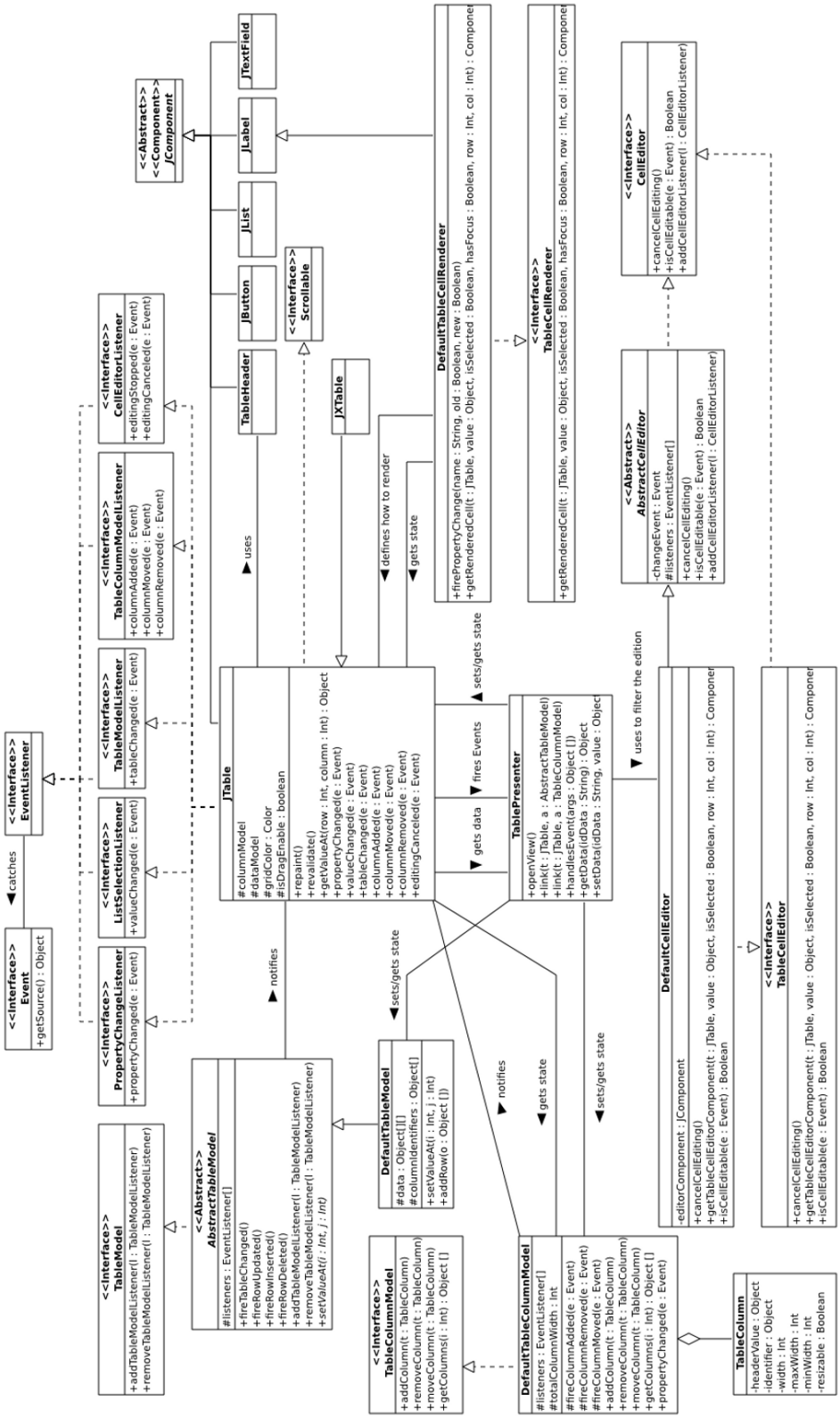


FIGURE E.6 – Swing / Modèle-Vue-Présentateur

Annexe F

Questionnaire de l'expérience

ÉCOLE
POLYTECHNIQUE
MONTRÉAL

Experiment - Subject # 1

Ptitdej

De Smet Benoît

Lempereur Lorent

Internship supervisor
Guéhéneuc Yann-Gaël

December 14, 2010

1 Experimental Procedure

During this experiment you will answer a set of questions concerning the maintenability of program represented by a UML class diagram. You will have to analyse two diagrams and answer 6 questions on each one. This experiment can last on average one hour. Please do not read/answer a question before the previous one.

Your eyes' movements will be recorded by an eye-tracker. The data collected during this experiment is totally **anonymous**. You can leave the experiment at any time for any reason without penalty of any kind. If you have any kind of questions do not hesitate to call us! :)
The experiment will follow the following steps:

1) We will calibrate the eye-tracker and we will show you a base picture and you will have to follow the line shown on the screen with your eyes. This step is only used to ensure that the eye-tracker is correctly calibrated.

2) Read the first question, then press `ENTER` when you are ready to proceed to start. It is **very important that you understood it right**, call us if you have a problem!

3) A diagram shows up on the screen. The question is visible on the top of the diagram. You have to think about the answer without getting your eyes away from the screen.

4) When you have an answer, press `ENTER` and write it on the paper. The diagram will still be on the screen, **use it only as support**.

5) When you have answered the question press `ENTER` to go to the next question.

6) A black screen with a cross will appear on the screen, please fix the cross and press `ENTER` when you are ready to proceed to the next screen.

7) Once all the questions for a diagram are answered, you will have to fill an short evaluation.

2 Good to Know!

In this section, we will give you some important things you have to know before the beginning of the experiment.

- Most of the getters and setters were removed from the diagram for clarity. Please note that **any field of a class can be accessed and modified by the methods of another class through getters and setters IF AND ONLY IF there is a `getState` association between them**.
- A class cannot interact with another if there is no association between them. Please pay attention to the direction of the associations.
- If the answer of a question is a method, please also write the class that declares it. Associations can be also used as methods.
- It is not necessary to specify the arguments of a method (polymorphism is not significantly used in the diagrams).
- An `Event` is **only** triggered by the action made by a **user**!

- The term `state` in (*gets/sets state*) defines the methods relative to the obtention/modification of the attributes of a class, **nothing else**.
- You can answer in French or in English.
- All abstract classes have the stereotype << *Abstract* >> for readability.
- When you are asked to describe a **logical path**, you must describe the different "calls" (methods or associations) between the classes as in the following example:

  Question: suppose that a seller wants to know the price of a specified order, describe the logical path of the information.

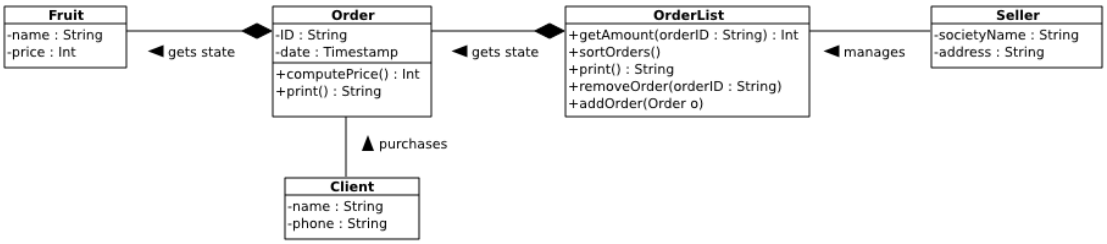


Figure 1: A simple diagram

Answer: The seller calls `getAmount(...)` of `OrderList`
then `OrderList` calls `computePrice()` of `Orders`
then `Order` gets state of `Fruit` *or* `Order` calls `getPrice()` of `Fruit`.

3 Diagram 1 – Swing

3.1 Context

Swing is part of Sun Microsystems' Java Foundation Classes and is an API for providing a graphical user interface (GUI) for Java programs. The diagram used in this section focuses on the *JTable*. A JTable is used to display and edit two-dimensional tables of cells.[1]

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	false
John	Doe	Rowing	3	true
Sue	Black	Knitting	2	false
Jane	White	Speed reading	20	true
Joe	Brown	Pool	10	false

Figure 2: A simple JTable

[1] Example from http://download.oracle.com/javase/tutorial/uiswing/components/table.html

3.2 Tasks

1) In which class(es) are the table data stored?

Answer:

2) Once the corresponding event is fired, which two concrete methods in two distinct classes are in charge of moving columns (as illustrated in FIGURE 3)?

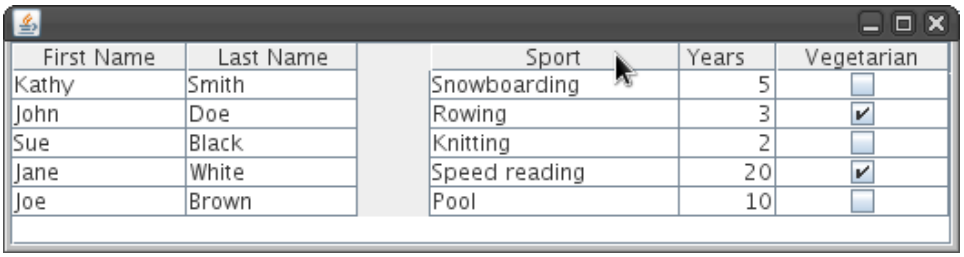


Figure 3: Moving a column in a JTable

Answer:

3) If you want to put a picture as a background for a cell, which class would you edit?

Answer:

4) Let us suppose that one instance of a `JTable` and one of `JXTable` exist and use the same `DefaultTableModel` instance and that the `JXTable` modifies a value in the `DefaultTableModel` instance. The method called when a user validates her input is *editingStopped(e : Event)* of the interface `CellEditorListener`. Describe the logical path until the *repaint* of the `JTable`.

Answer:

5) Let us assume that a table only includes integers and that you want that the value shown in the table always be the double (multiplication by 2) of the real data stored in the table data model. Which method would you modify or overload? Example: if a cell contains 7, the shown value is 14. **The data must not be modified!** The table is not editable, you must ignore this aspect.

Answer:

6) If you want to enable a user to choose his unit system (cm or inch via a popup menu) on a table presenting the dimensions of some pieces of furniture, what method would you overload to allow this new behaviour (as illustrated in FIGURE 4)? **The data must not be changed and you do not have to manage the pop up menu** Please focus on the unit systems.

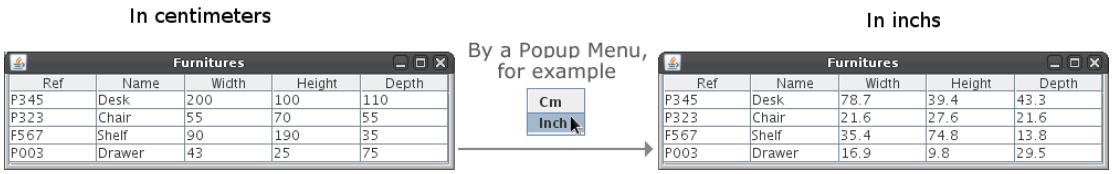


Figure 4: Two systems on same data

Answer:

3.3 Evaluation

This section is used to evaluate your effort about the Swing's questions.

Mental Demand: How much mental demand and perceptual activity was required (e.g thinking, deciding, calculating, remembering, looking, searching etc)? Was the tasks easy or demanding, simple or complex, exacting or forgiving?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Physical demand: How much physical activity was required e.g. pushing, pulling, turning, controlling, activating etc? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Temporal demand: How much time pressure did you feel due to the rate or pace at which the tasks elements occurred? Was the pace slow and leisurely or rapid and frantic?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Effort: How hard did you have to work (mentally and physically) to accomplish your level of performance?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Performance: How successful do you think you were in accomplishing the goals of the task set by the analyst (or yourself)? How satisfied were you with your performance in accomplishing these goals?

Poor Middle Good
O O O O O O O O O O O O O O O O O O O O

Frustration level: How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?

Low Middle High

O O O O O O O O O O O O O O O O O O O O

4 Diagram 2 - JFreeChart

4.1 Context

JFreeChart is a free Java chart API that makes it easier for developers to display charts in their applications[2].

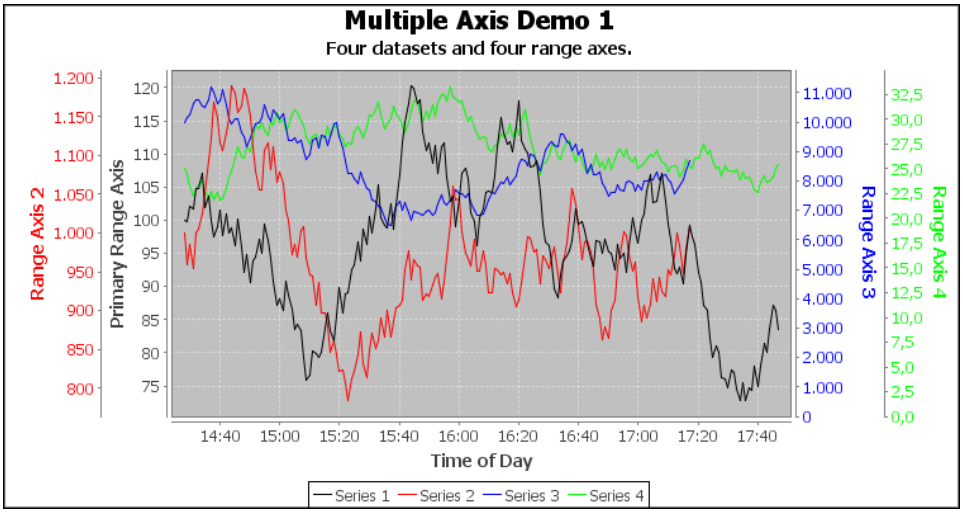


Figure 5: A JFree chart demo

[2] http://www.jfree.org/jfreechart/samples.html

4.2 Tasks

1) In which class(es) are the chart data stored?

Answer:

2) Let us suppose that the data of a pie chart are modified by the execution of the program (**not by the user, so not an event**). Describe the logical path of the data until the chart is updated.

Answer:

3) If you wanted to add a feature to zoom with the *mousewheel*, which classe(s) would you edit?

Answer:

4) Which class should you edit to add the possibility to set a video as a background for a graphic?

Answer:

5) Let us assume two users of the same pie chart on identical data. One user is administrator and the other is a simple user. The admin has access to an interactive interface allowing some actions on the chart (zooming or editing the colours and the data for examples. The simple user has only access to a static view (**no event** are possible, the user cannot interact with the chart). Which classe(s) should you extend/duplicate/edit to handle this new behaviour (the separation of the interface usage rights) and how?

Answer:

6) If a program must handle 3 monetary systems (€, US$, CAN$) and if a pie chart shows the average wage of some employees, what method would you overload for each monetary system (getters and setters included)? Your answer should work with the *Category Chart* and the *Value Chart* too. **The data are already set in the diagram (do not modify it) and assume that the user is already handled by the diagram.** Please focus on the monetary systems.

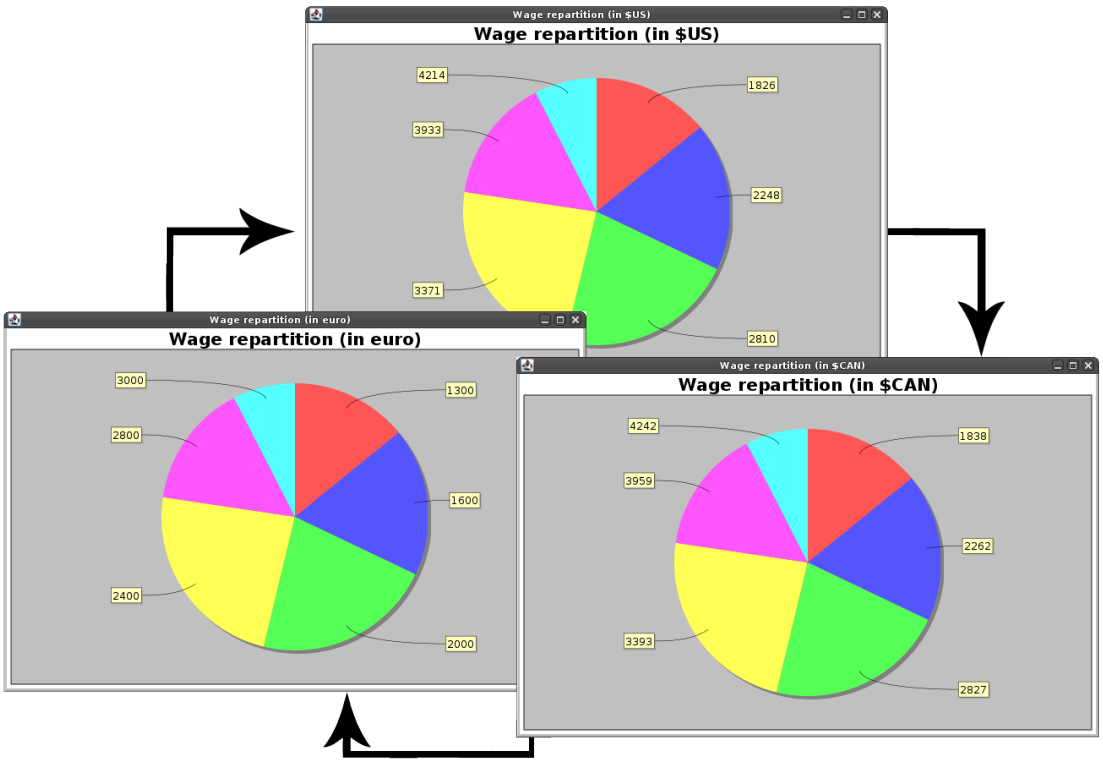


Figure 6: Three systems on same data

Answer:

4.3 Evaluation

This section is used to evaluate your effort about the JFreeChart's questions.

Mental Demand: How much mental demand and perceptual activity was required (e.g thinking, deciding, calculating, remembering, looking, searching etc)? Was the tasks easy or demanding, simple or complex, exacting or forgiving?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Physical demand: How much physical activity was required e.g. pushing, pulling, turning, controlling, activating etc? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Temporal demand: How much time pressure did you feel due to the rate or pace at which the tasks elements occurred? Was the pace slow and leisurely or rapid and frantic?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Effort: How hard did you have to work (mentally and physically) to accomplish your level of performance?

Low Middle High
O O O O O O O O O O O O O O O O O O O O

Performance: How successful do you think you were in accomplishing the goals of the task set by the analyst (or yourself)? How satisfied were you with your performance in accomplishing these goals?

Poor Middle Good
O O O O O O O O O O O O O O O O O O O O

Frustration Level: How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?

Low Middle High

O O O O O O O O O O O O O O O O O O O O O

ANNEXE G

Questionnaire post-expérimental

G.1 Miscellaneous

1. Subject number ?
2. You are...
   - Male
   - Female
3. What is your higher grade ?
   - Bac
   - Master
   - PhD
   - Other

G.2 MVC

1. What does M.V.C. stand for ?
2. In a M.V.C. architecture, could there be more than one V ?
   - Yes
   - No
   - Unknown
3. In a MVC architecture if you need to handle the actions of the users, where would you put the related code ?
   - In the M
   - In the V
   - In the C
   - Unknown
   - Other
4. What design pattern is mostly used to make the M able to notify the V about his changes ?

G.3 UML

1. How many experience in UML do you have ?
2. In a UML class diagram, a protected attribute is preceded by.....
   - +
   - #

- -
- $\sim$
- Unknown

3. In an UML class diagram, an generalisation / specialisation relationship is represented by...
   - A square
   - A diamond
   - A triangle
   - A cercle
   - Nothing
   - Unknown
4. Which cardinality is used to represent a relation "at least 1" ?
5. What is the name of this relationship ? (Figure G.1)
   - Composition
   - Aggregation
   - Realisation / Implementation
   - Generalization
   - Unknown
   - Other
6. One of these relations is NOT a classification, which one ?
   - Généralisation
   - Association
   - Spécialisation
   - Unknown
7. An attribute is...
   - A data having a value for each object
   - A data having a value for each class
   - A domain of values
   - Unknown
8. Which format is used to define a abstract class ?
   - Bold
   - Italic
   - Underlined
   - Unknown
9. If a specialization relationship exists between two classes, then.....
   - The subclass inherit attributes from the parent
   - The subclass inherits attributes from other parent's subclasses
   - The subclass has to redefine all parent's methods
   - Unknown

G.4 Personnal experience

1. Have you ever handled (add, delete, modify or maintained) a UML class diagram system that has more than 20 classes ?
   - Yes
   - No

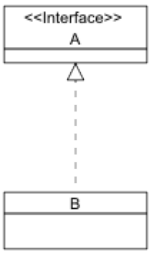


FIGURE G.1 – Illustration d'une question post-expérimentale

G.5 Comments

1. If you have any suggestions / critics about the experiment please use this field